Botnet detection based on traffic behavior analysis and flow intervals

Article *in* Computers & Security · November 2013

DOI: 10.1016/j.cose.2013.04.007

Issa Traore
University of Victoria
147 PUBLICATIONS **1,784** CITATIONS
SEE PROFILE

Bassam Sayed
University of Victoria
10 PUBLICATIONS **414** CITATIONS
SEE PROFILE

Wei Lu
Keene State College
48 PUBLICATIONS **2,084** CITATIONS
SEE PROFILE

Sherif Saad
University of Windsor
26 PUBLICATIONS **487** CITATIONS
SEE PROFILE

Available online at www.sciencedirect.com

SciVerse ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Botnet detection based on traffic behavior analysis and flow intervals

David Zhao[a], *Issa Traore*[a,*], *Bassam Sayed*[a], *Wei Lu*[b], *Sherif Saad*[a], *Ali Ghorbani*[c], *Dan Garant*[b]

[a] *Department of Electrical and Computer Engineering, University Of Victoria, Victoria, BC, Canada V8W 3P6*
[b] *Keene State College, NH, USA*
[c] *Faculty of Computer Science, University of New Brunswick, NB, Canada*

ARTICLE INFO

Article history:
Received 27 November 2012
Received in revised form
22 March 2013
Accepted 24 April 2013

Keywords:
Botnet
Intrusion detection
Traffic behavior analysis
Network flows
Machine learning

ABSTRACT

Botnets represent one of the most serious cybersecurity threats faced by organizations today. Botnets have been used as the main vector in carrying many cyber crimes reported in the recent news. While a significant amount of research has been accomplished on botnet analysis and detection, several challenges remain unaddressed, such as the ability to design detectors which can cope with new forms of botnets. In this paper, we propose a new approach to detect botnet activity based on traffic behavior analysis by classifying network traffic behavior using machine learning. Traffic behavior analysis methods do not depend on the packets payload, which means that they can work with encrypted network communication protocols. Network traffic information can usually be easily retrieved from various network devices without affecting significantly network performance or service availability. We study the feasibility of detecting botnet activity without having seen a complete network flow by classifying behavior based on time intervals. Using existing datasets, we show experimentally that it is possible to identify the presence of existing and unknown botnets activity with high accuracy even with very small time windows.

1. Introduction

A botnet is a collection of computers connected to the Internet which have been compromised and are being controlled remotely by an intruder (the botmaster) via malicious software called bots (Rajab et al., 2006; Feily et al., 2009; Al-Duwairi and Al-Ebbini, 2010). Botnets are created for many different reasons such as conducting a distributed denial of service (DDOS), spreading spam, conducting click-fraud scams, stealing personal user information (e.g. credit card numbers, social security numbers), or taking advantage of the powerful computational resources offered by the bots to carry some distributed computing tasks. A key aspect of any botnet is the communication architecture. Bots interact over legitimate communication channels. Internet Relay Chat (IRC) used to be the most prevalent communication scheme among traditional botnets until the early 2000s. After infection, the bot will locate and connect with an IRC server. The bot master will use established IRC command and control (C&C) channels to communicate and control the bots. The bot master will try to keep the bots under control as long as possible. From time to time the bots will connect to the bot master to get new

[*] *Corresponding author*. Tel.: +1 250 721 8697.

E-mail addresses: davidzhao@ieee.org (D. Zhao), itraore@ece.uvic.ca (I. Traore), bassam@ece.uvic.ca (B. Sayed), wlu@keene.edu (W. Lu), shsaad@ece.uvic.ca (S. Saad), ghorbani@unb.ca (A. Ghorbani), dgarant@ksc.keene.edu (D. Garant).

URL: http://www.isot.ece.uvic.ca

0167-4048/$ – see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cose.2013.04.007

instructions and update their behavior. However, IRC-based bots are vulnerable because they are based on highly centralized architectures: the entire botnet can be disrupted by simply shutting down the IRC 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.

New bots evolved around the mid-2000 that 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 (Grizzard et al., 2007; Holz et al., 2008). However, P2P botnets have their own limitations rooted mainly in the higher latency underlying the command and control transmission and the impact of such latency on bots synchronization.

C&C schemes based on HTTP traffic is a more recent method for botnets which gained interest lately. 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.

Recent developments also include the investigation of hybrid models, which address the limitations of the above mentioned botnets architectures by employing characteristics from both centralized and P2P structures (Wang et al., 2007).

According to Leonard et al. (2009), a botnet's lifecycle involves four phases: formation, C&C, attack and post-attack. The formation phase is the initial phase where an attacker exploits a known vulnerability for a target system, infects the victim machine, and 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. The bot will attempt 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 attack phase is noted as a phase in the botnet lifecycle when the bot is actively performing malicious activities based on received instructions. The post-attack phase is the last phase of the botnet lifecycle, during which bots are commanded to update their binaries, typically to defend against new attacks or to improve their functionality.

Many existing botnet detection techniques rely on detecting bot activity during the attack phase or initial formation 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 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. 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 the formation phase, 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.

Another limitation of most existing detection techniques is that they are good at detecting only known forms of botnets technologies. They tend to be ineffective when faced with new forms of botnets. We assess the capability of our detector against new botnet attacks by testing it with traffic samples generated by two new botnet instances, yielding very encouraging performance results.

The rest of the paper is structured as follows. Section 2 summarizes and discusses related work on botnet detection. Section 3 introduces our proposed detection model. Section 4 outlines the experimental evaluation settings and results for our detector. Section 5 describes instances of new botnets used in our evaluation and then presents corresponding evaluation results. Section 6 makes some concluding remarks and outlines our future work.

2. Related work

Although a significant amount of literature has been produced on botnet detection, botnet detection approaches using flow analysis techniques have only emerged in the last few years (Gao et al., 2009) and of these most examine flows in their entirety instead of smaller time intervals.

Masud et al. (2008) propose a P2P botnet detection approach by treating botnet traffic as stream data. By considering that such data is characterized by infinite length and drifting concept, the authors propose a multi-chunk, multi-level ensemble classification algorithm, in which trained classifiers are stored instead of historical data. Empirical evaluation of the proposed approach using both botnet traffic and simulated data yields better detection accuracies than existing stream data classification techniques.

Gu et al. proposed successively two botnet detection frameworks named BotHunter (Gu et al., 2007) and BotMiner (Gu et al., 2008b). BotHunter (Gu et al., 2007) 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 (Gu et al., 2008b) 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 first clusters similar communication activities in the so-called C-plane (for C&C communication traffic). 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 clusters in the so-called A-Plane (for activity traffic) 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. (2010) 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%. 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 and Rouhani (2012) 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 bots lifecycle. Additionally, similarity and group behavior detection strategies rely 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. In this context, Giroire et al. (2009) focus their work on detecting botnet C&C communications on an end host. Their working assumption is that a recruited host needs to keep in touch with the bot master to remain relevant. This is characterized by frequent communications between the bot master and the bot. Such communications exhibit some temporal regularity even though they may be spread over irregular large time periods. Such regularity is captured by designing a detector that monitors outgoing traffic for a given user and identifies malicious destinations visited with some temporal regularity. Malicious destinations are distinguished from normal ones by building destination IP white lists referred to as atoms by the authors. To capture the temporal regularity of visited destinations, a feature referred to as persistence is introduced. Detection of botnet C&C channels is carried out by identifying non-white listed destinations with high persistence using thresholding methods. Evaluation of the proposed approach using real network traces yields low false positive rate.

Wurzinger et al. (2009) propose an approach for detecting individual hosts in a monitored network that are members of a botnet by observing and correlating commands and responses in network traces. More specifically, the proposed technique first identifies responses in network traces and then analyzes the preceding traffic to locate the corresponding commands. Then, based on the identified commands and responses patterns, detection models for similar activity in the network are built and used to establish the presence of botnet. The generated models are specific to particular bot families. To evaluate the proposed framework, detection models for 18 bots including IRC, HTTP, and P2P-based were automatically generated, yielding low false positive rate on network traces generated using publicly available bot samples.

Livadas et al. (2006) 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. 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%–20% and a false positive rate of 30%–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. (2009) presented a detection approach of peer-to-peer based botnets (specifically Storm botnet and its variants using the Kademlia based protocol) by observing the stability of control flows in initial time intervals of 10 min. 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.

In an earlier version of the current paper, presented at the 2012 IFIP International Information Security and Privacy Conference (SEC 2012), we investigated botnet detection based on flow intervals using two different machine learning techniques, namely decision tree and Naive Bayes classifiers (Zhao et al., 2012). While the evaluation of the proposed approach yields encouraging results, it does not account for novelty detection. Novelty detection refers to the ability of a machine learning technique to identify new or unknown data that it has not been trained with and was unaware of during the model construction. The current paper extends the work presented in Zhao et al., (2012) by conducting an assessment of the ability of the proposed approach to detect new botnets that were not involved in training the detector.

3. Flow analysis model

In this section, we discuss the benefit of traffic analysis in botnet detection, and then introduce our flow analysis approach and model which is based on traffic analysis.

3.1. *Traffic analysis*

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 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.

3.2. *Approach overview*

The uniformity of botnet communications and botnet behavior is well known and has been exploited by various architectures toward their detection (Gu et al., 2008b; Yu et al., 2010; Villamarn-Salomn and Brustoloni, 2009; Li et al., 2007). 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. (2008a) detect individual bots by drawing a spatial–temporal correlation in the responses of bots to a specific command. 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 for detecting peer-to-peer traffic based on a set of network flow attributes (The Honeynet Project). 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 (e.g. 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 (Grizzard et al., 2007; Sinclair et al., 2009; Szab et al., 2008). 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 (Sperotto et al., 2010). 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.

3.3. *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 to build our detector. 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. 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 they query their 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 bots 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.

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

3.4. *Classification model*

Many traditional techniques for detecting botnets and malicious network activity consist of rule based or signature based detection. Such techniques typically utilize existing network intrusion detection systems (IDS) to correlate the presence of specific activity (such as sending spam, scanning a port, or even performing a known exploit) with the existence of a malicious agent on a given system or network. The generation of rules for such systems is often manual and performed on a case by case basis as new malware and malicious techniques are discovered. As malicious software proliferates, such manually generated rules are becoming increasingly impractical, and an automated alternative is desirable. Machine learning techniques, similar to those used in automated spam filtering, allow detection systems to adapt quickly to rapidly changing malware and in addition allow for an automated method for discovering unique patterns and properties that may be exhibited by malware. 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 investigated several machine learning techniques for botnet detection through network behavior analysis, including, Bayesian Network, Neural Networks, Support Vector Machine, Gaussian and Nearest Neighbor Classifiers in

Saad et al. (2011), and Naive Bayes and Decision Tree in Zhao et al. (2012).

Based on the above criteria, we have selected decision tree classifier, which is a popular machine learning for our detection framework. Automated decision tree classifiers uses a decision tree as a predictive model, taking in a series of observations and generating a value for those given inputs. Leaf nodes in a decision tree represent a class or a node linked to two or more subtrees called a test node. At a test node, some outcome is computed based on the attribute values of an instance, where each possible outcome leads to one of the subtrees rooted at the node. Automated algorithms which construct decision trees are prone to over-fitting data, creating many branches in the tree in order to capture very specific and complex scenarios in a given training set. Pruning algorithms are therefore used on decision trees in order to trim the size of the tree and reduce its complexity. For any pruning algorithm, the goal is to determine the amount of error that a decision tree would suffer before and after each pruning stage, and then decide how to prune to avoid error. The estimate of the possible error is determined using the error estimate given by:

$$E = \frac{e+1}{N+m}$$

where e represents misclassified examples at a given node, N represents the number of examples that would reach the node, and m represents the set of all training examples.

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. The resulting tree may be seen in Fig. 1. In Fig. 1, a single branch of the tree is given in graphical form. In reality, the actual tree spans several thousand nodes and is difficult to represent on a single sheet of paper due to its graphical complexity.

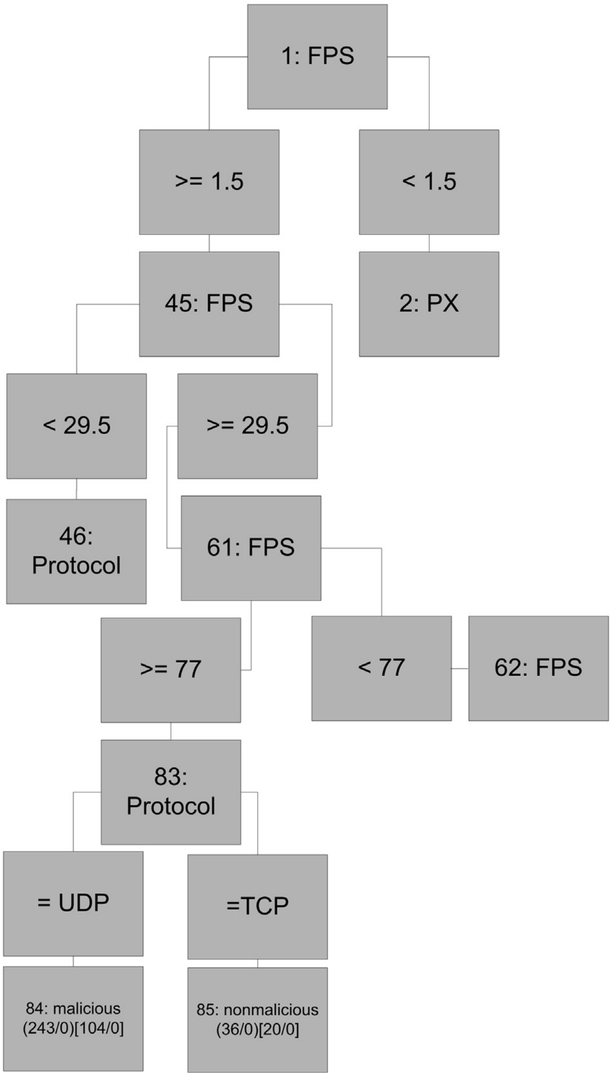


Fig. 1 – Resulting decision tree produced by our algorithm on sample data. A decision path points to either a non-malicious traffic or a malicious traffic sample. Each node indicates a decision point, for example, the root indicates that if FPS is <1.5, then the PX value should be examined.

4. Experimental evaluation

In this section, we present our approach in evaluating our botnet detection approach. In the first subsection, we introduce our test dataset and briefly describe our data generation methods. In subsequent sections, we discuss the results of our test runs on the dataset.

4.1. *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 (The Honeynet Project). To represent non-malicious, everyday usage traffic, we incorporated two different datasets, one from the Traffic Lab at Ericsson Research in Hungary (Szab et al., 2008) and the other from the Lawrence Berkeley National Laboratory (LBNL) (2005). 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

Table 2 – LBNL datasets general information.

	D_0	D_1	D_2	D_3	D_4
Date	4/10/04	15/12/04	16/12/04	6/1/05	7/1/05
Duration	10 min	1 h	1 h	1 h	1 h
Subnets	22	22	22	18	18
Hosts	2531	2102	2088	1561	1558
Packets	18 M	65 M	28 M	22 M	28 M

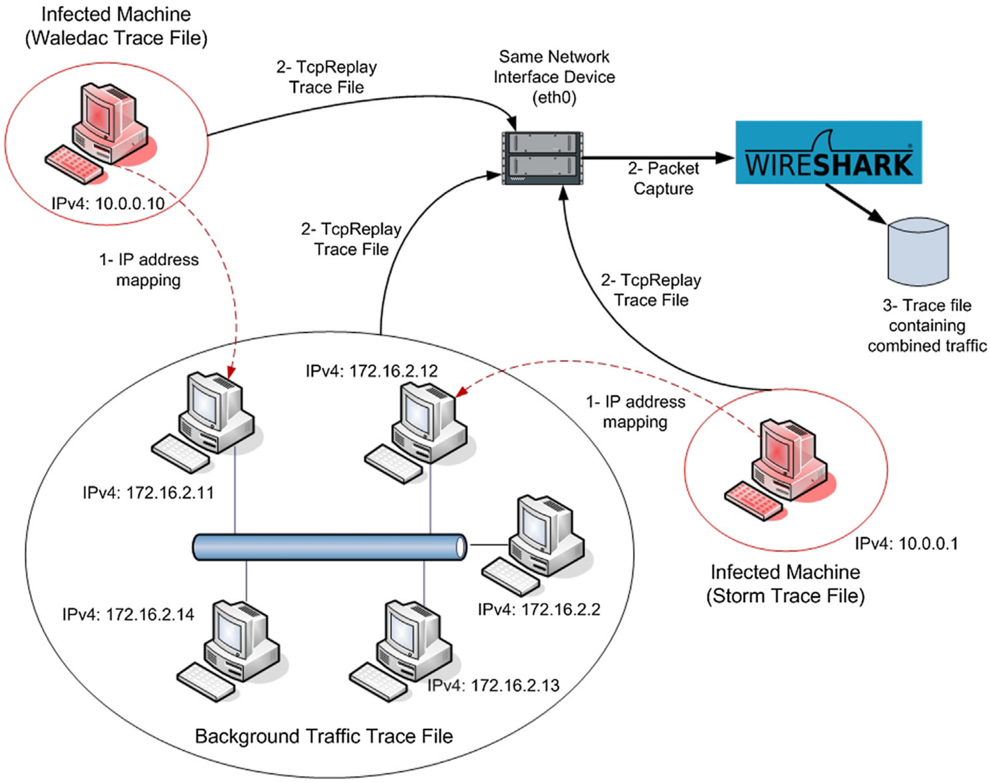


Fig. 2 – Dataset merging process.

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

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 Fig. 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 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 Fig. 2 with both malicious and non-malicious traffic originating from the same machines.[1]

4.2. *Model evaluation results*

We implemented our framework in Java and utilized the popular Weka machine learning framework and libraries for our classification algorithms (Witten et al., 1999). 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 s 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

[1] The dataset is publicly available and can be obtained at http://www.uvic.ca/engineering/ece/isot/datasets/index.php.

Storm or Waledac 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 the decision tree classifier are listed in Table 3. The resulting detection values are an average of the results of the ten runs.

As can be seen in the above table, the decision tree classifier produced very high (above 90%) detection rates with a very low false positive rate. These results indicate that there are indeed unique characteristics of the evaluation botnets when compared to everyday network traffic.

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 4, to be used as an optimized subset that may improve our performance without producing a large reduction in accuracy.

Table 5 lists 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 REPTree classifier with the reduced set of attributes decreased slightly due to an increased false positive rate, compared to the case with the full attribute set. In terms of performance, reducing the number of attributes allowed the RepTree classifier to classify all attribute vectors in 33% of the original time. Table 6 shows the actual times for classifying all attribute vectors by the classifier on both the full attribute set and the reduced set.

With the above results, we may conclude that the decision tree classifier is 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. Figs. 3 and 4 show the effects of the time window size on the true and false positive rates for malicious flow detection. We notice a sharp decrease in the False positive rate up to $T = 60$ s where it starts stabilizing and then saturates around $T = 180$ s and beyond. In contrast, we notice a slightly sharper increase in the true positive rate as the time window increases up to $T = 60$ s, from where it starts stabilizing and then progressively saturates at $T = 180$ s and beyond. The optimal operating point can be selected by making a trade-off between time window, true positive rate, and false positive rate, which should be kept low, high, and low, respectively. In our experiments, the best results were obtained when $T = 180$ s, though very good results were obtained for all four time window sizes (10, 60, 180 and 300 s).

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 (Gu et al., 2007) 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 (Snort, 1999). 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

Table 4 – Attribute subset from CFS subset evaluation.

Attribute	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	Number of flows per address/total flows

Table 3 – Detection rate of REPTree classifier ($T = 300$ s).

	True positive	False positive
Malicious	98.3%	0.01%
Non-malicious	99.9%	1.7%

Table 5 – Detection rate of REPTree classifier with reduced subset ($T = 300$ s).

	True positive	False positive
Malicious	98.1%	2.1%
Non-malicious	97.9%	1.9%

Table 6 – Classifier performance (average training time).

Classifier	Time (seconds)
RepTree	29.4
RepTree (subset)	8.58

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. Novel botnet detection

We discuss, in this section, the implementation of our detector and its use for new botnet detection.

5.1. *Detector implementation*

Our algorithm may be used for both offline detection as well as live detection in a production environment. In order to perform further evaluations as well as demonstrate the efficacy of the system, we have implemented a web based detection system which may be deployed in a live setting. Fig. 5 shows the system as it is viewed on a web browser. The live version of the botnet detection system was written in Java, using Google Web Toolkit (GWT) for the presentation layer.

The user workflow of the system starts by uploading training files, i.e., pcap files containing malicious and non-malicious traffic data. Alternatively, the user can specify a device and an interface to capture packets on for a given time in order to generate a file that way. The uploaded or captured data will be used to generate a signature which will be used for detection. For live detection, the user must select a device and an interface to capture packets and perform live detection on. For offline detection, the user must upload and specify traffic files that will be processed by the detector to check for the presence of botnets.

5.2. *Novel botnets*

We consider two classes of HTTP-based botnet frameworks which we are interested in detecting as sample new threats, namely, the Weasel and BlackEnergy botnets. The first class, which Weasel belongs to, uses a fully-encrypted communication channel and a dynamic action set. The second class does not employ encryption, and uses a static set of configurable actions. In our analysis, we use BlackEnergy as a specimen belonging to the second framework category. Weasel is a new botnet designed by some of the authors (of this paper) for research purpose, while BlackEnergy is a relatively recent botnet whose inner-working has been exposed through the work of Jose Nazario (2007). We consider these two botnets as sample new botnets, firstly, because of their novelty, and secondly, because none of them informed the design of our detector, nor were they involved in our training dataset.

5.2.1. *Weasel*

The Weasel botnet was designed with the intention of providing an open-source platform for analysis of botnet traffic. The focus of the development effort was in making the communication channel realistic and secure, and the primary design concern is implementing secure communication between bots and the botmaster.

Under the Weasel framework, there are three distinct modules as depicted by Fig. 6. The bot module can be viewed

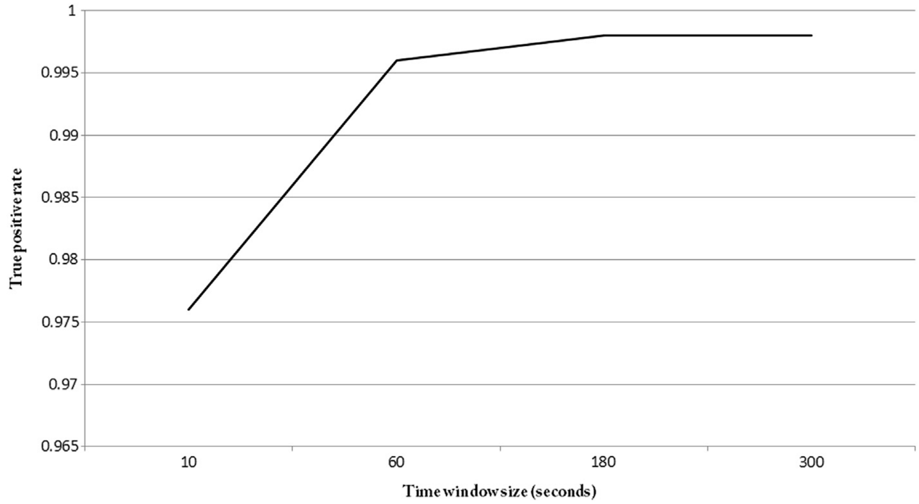


Fig. 3 – Effects of time window size on true positive rate for REPTree classifier.

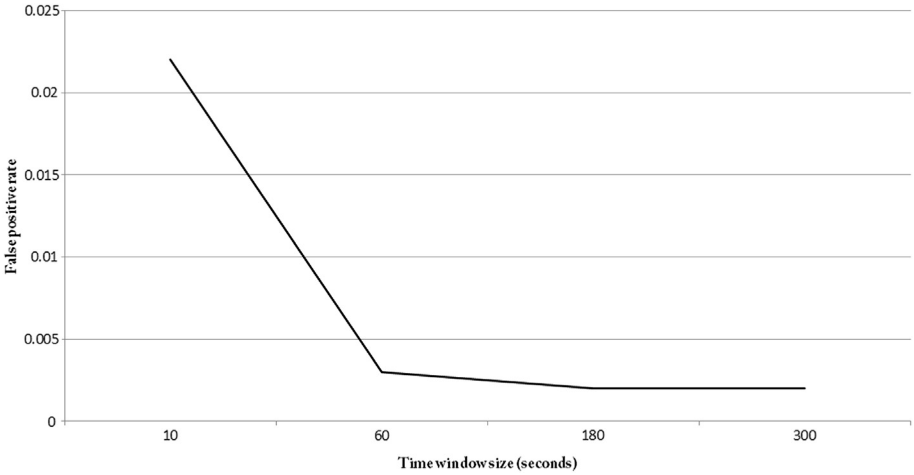
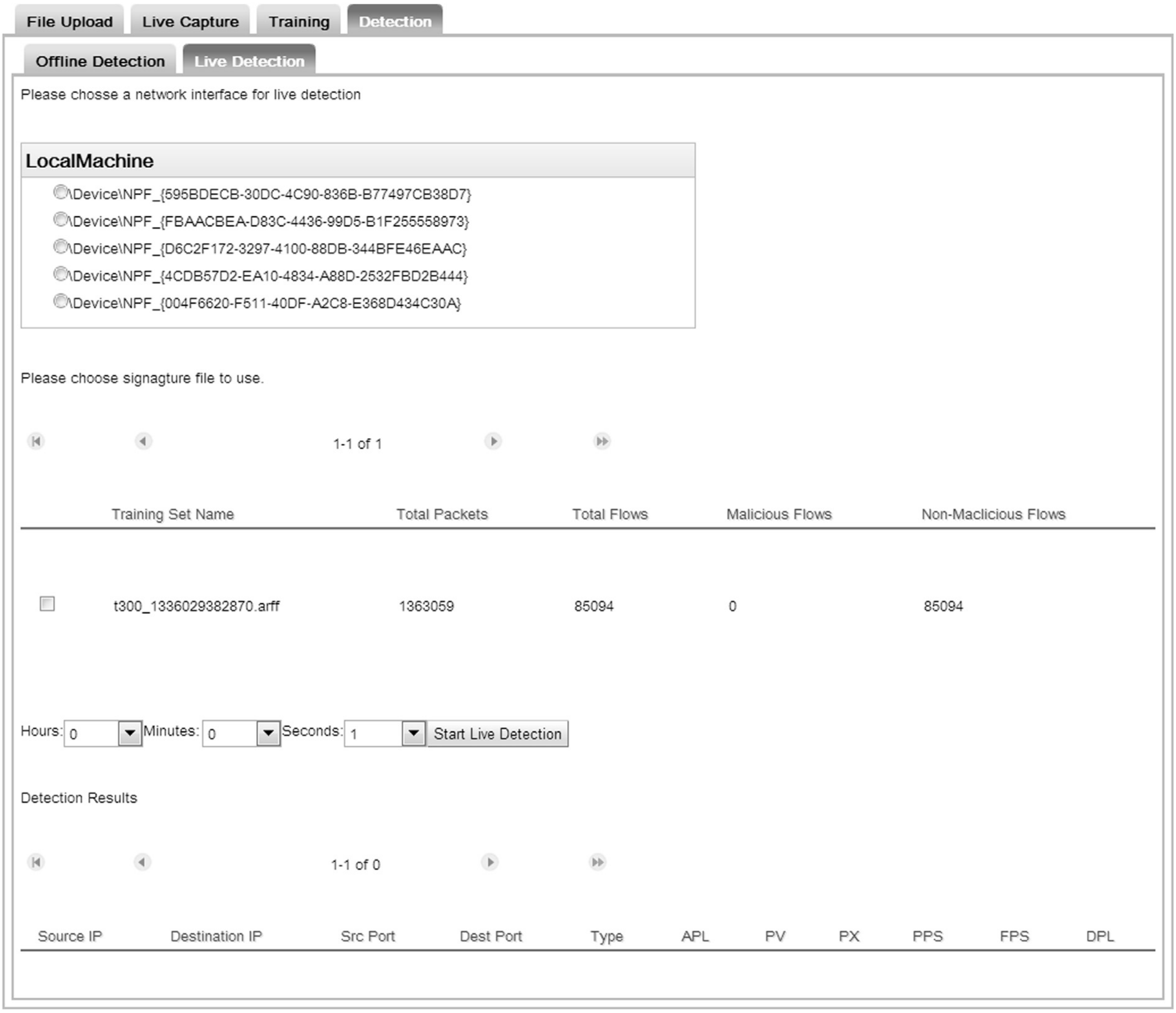


Fig. 4 – Effects of time window size on false positive rate for REPTree classifier.

as a server which listens for socket connections from the command module. The command subsystem is the only true client component in the botnet system. It is invoked on demand as the botmaster wishes to send commands. Both the command and bot modules interact with a RESTful Python web service, which contains functionality for updating bot status and maintaining a history of sent commands. We can consider this module to be the control, and this is what truly



Please chosse a network interface for live detection

LocalMachine

○\Device\NPF_{595BDECB-30DC-4C90-836B-B77497CB38D7}
○\Device\NPF_{FBAACBEA-D83C-4436-99D5-B1F255558973}
○\Device\NPF_{D6C2F172-3297-4100-88DB-344BFE46EAAC}
○\Device\NPF_{4CDB57D2-EA10-4834-A88D-2532FBD2B444}
○\Device\NPF_{004F6620-F511-40DF-A2C8-E368D434C30A}

Please choose signagture file to use.

1-1 of 1

	Training Set Name	Total Packets	Total Flows	Malicious Flows	Non-Maclicious Flows
□	t300_1336029382870.arff	1363059	85094	0	85094

Hours: 0 Minutes: 0 Seconds: 1

Detection Results

1-1 of 0

Source IP	Destination IP	Src Port	Dest Port	Type	APL	PV	PX	PPS	FPS	DPL

Fig. 5 – Screenshot of online detector.

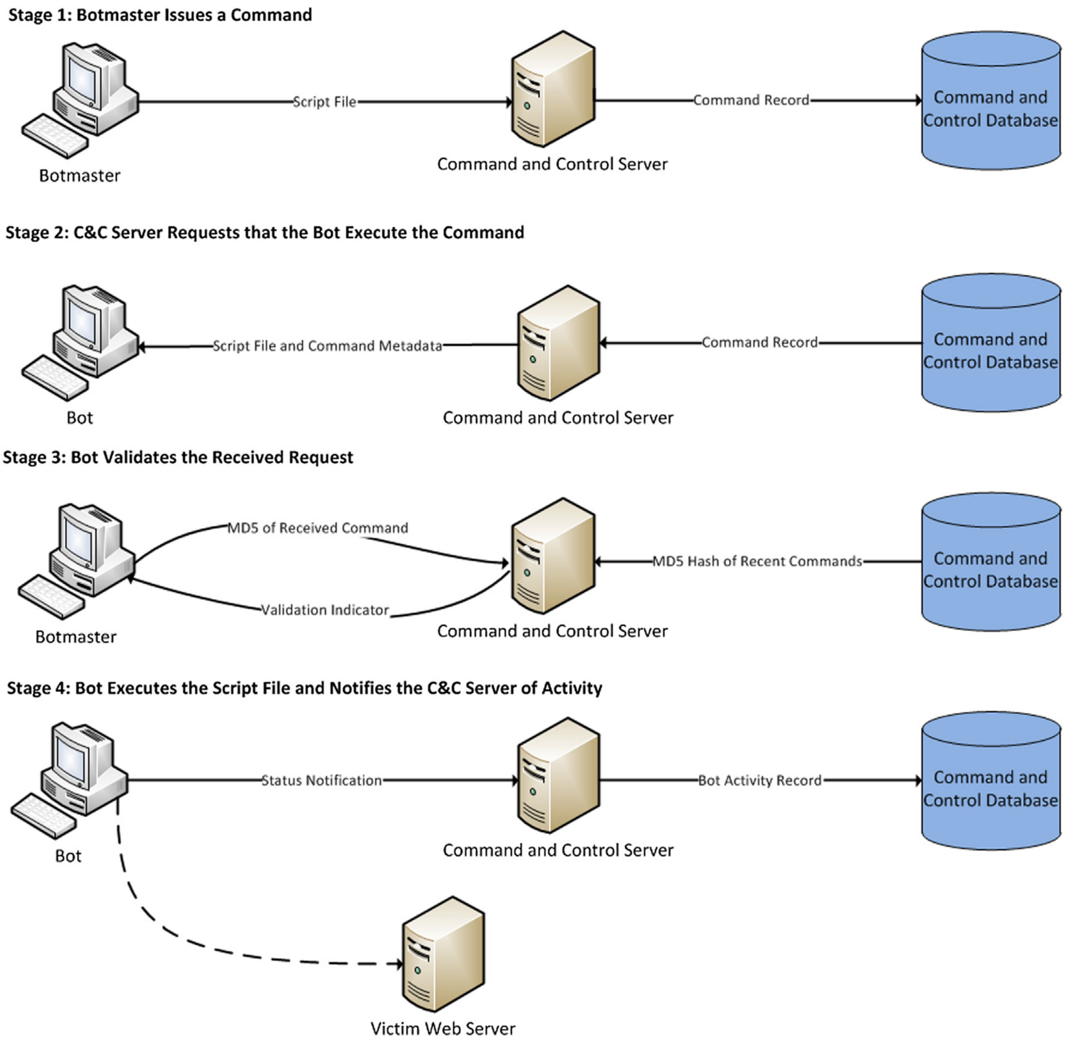

Stage 1: Botmaster Issues a Command
Botmaster
Script File
Command and Control Server
Command Record
Command and Control Database
Stage 2: C&C Server Requests that the Bot Execute the Command
Bot
Script File and Command Metadata
Command and Control Server
Command Record
Command and Control Database
Stage 3: Bot Validates the Received Request
Botmaster
MD5 of Received Command
Validation Indicator
Command and Control Server
MD5 Hash of Recent Commands
Command and Control Database
Stage 4: Bot Executes the Script File and Notifies the C&C Server of Activity
Bot
Status Notification
Command and Control Server
Bot Activity Record
Command and Control Database
Victim Web Server

Fig. 6 – Weasel communication overview.

centralizes the Weasel botnet. In order to coordinate the distributed activity between the separate modules, Weasel makes use of a shared PostgreSQL database. The command & control modules have direct access to this database, whereas the bot has indirect access through the web service. This database maintains entities for bot status tracking, command history, command participation, and user authentication. Weasel provides an HTTPS-encrypted web interface as part of the control module containing a few basic views into the current state of the botnet. Specifically, this interface provides read-only access to bot status information, command history, and user accounts. Access to this interface is controlled with the C&C database account table, on the basis of client IP address and a password. This layer of security ensures that the sensitive information being transmitted within the bot network is visible to only those hosts which the botmaster designates. Driving the botnet communication model is a simple language shared by the botmaster and the bots. Weasel does not encode any knowledge of the type of commands that will be issued, instead, commands are written as Python script files which are transmitted to bots for execution. This implies that the Weasel botnet can be used for DDoS attacks, spamming, or even non-malicious purposes, such as distributed computing. Security in the bot to botmaster communication is accomplished using HTTPS encryption. Since the C&C server runs Apache, these modules can take advantage of the SSL engine Apache provides to wrap the service interface. When a bot starts up, executes a command from the botmaster, or shuts down, it issues an HTTP GET request to the C&C server which informs the server of the current activity. These activity notifications are sent so that the botmaster knows not to communicate with unresponsive bots, and also so that the botmaster is aware of newly established bots. To prevent hijacking of the botnet, bots perform basic message authentication & validation. Upon receipt of a command from the botmaster, the bot issues a validation request to the C&C server, containing the MD5 hash of the received content. The server scans the history of commands issued within the past 10 s, and if there is a match between the stored hash value and the sent hash value, the service indicates that the command is valid. This brief timeframe limitation prevents replay of communications.

5.2.2. *BlackEnergy*

BlackEnergy is a botnet designed and used for DDoS attacks. The command and control server uses a web interface built on PHP and MySQL to issue commands. In this analysis, we focus on the salient features and concepts of the BlackEnergy 1.7 kit. Much of our knowledge of BlackEnergy comes as a result of work done by Jose Nazario (2007) in analyzing this framework.

All communications in the BlackEnergy botnet are initiated from the bot, using Base64-encoded HTTP communication. The bot makes periodic HTTP GET requests to the C&C server. The server responds with the current command state, as defined in the C&C database. The bot reads and parses the HTTP response, and updates its action configuration accordingly. Fig. 7 illustrates a typical BlackEnergy communication sequence. BlackEnergy issues only a few commands, designed with the sole purpose of executing denial-of-service attacks. These commands include flooding a server using HTTP GET requests, ICMP ping flooding, and others. Each of these operations maps directly to a function encoded in the bot. These functions are parameterized by several configuration values and embedded in the C&C servers response to the bot. Specifically, BlackEnergy makes it possible to adjust request frequencies, packet sizes, spoofed addresses, and thread usage.

5.3. *Novelty evaluation*

We trained our live detector using an extended version of our dataset presented above in Section 4. In addition to the two previous malicious datasets containing Storm and Waledac botnets, we incorporated in our datasets the following two new malicious datasets:

1. A dataset involving a mix of Zeus botnet C&C and sample traffic from openpacket.org website.
2. A subset of the ISCX 2012 IDS dataset containing traffic from an IRC botnet described in (Shiravi et al.).

Table 7 – Novelty detection results without filtering.

	BlackEnergy	Weasel
Number of alerts	3154	3525
Detection rate	100%	100%
False positive rate	0%	82%

For testing, sample Weasel and BlackEnergy traffic were captured independently (on different networks from where the above training samples were collected). The BlackEnergy dataset consists of 19 MB of (entirely) malicious traffic generated by 3 bots controlled by a botmaster.

The Weasel dataset consists of 80 MB of traffic. The data was collected in an experiment designed so that the botmaster would issue a randomly selected command (from a pool of potential script files) to the bot every 4 h. The botnet was deployed on a public web server where the firewall (IPtables) was disabled allowing all traffic coming in and out. As a result, the collected data consists of a mixture of malicious and normal communications.

None of the Weasel and BlackEnergy data was used in designing or training the detector. We used by default a detection time window $T = 300$ s. Table 7 shows the results obtained by running the trained detector on each of the test datasets.

For BlackEnergy, 3154 alerts covering all the IP addresses in the dataset were generated. In other words, we successfully flagged all malicious machines as malicious. Since the dataset consists of only malicious traffic, this indicates that no false positive was generated in this case. For Weasel, a number of 3525 alerts were generated. Out of the 3525 alerts, 623 correctly flagged the bot or botmaster traffic as malicious which yields 100% detection rate. However, this means also that the remaining 2902 alerts were false positives, which corresponds to a false positive rate of 82%.

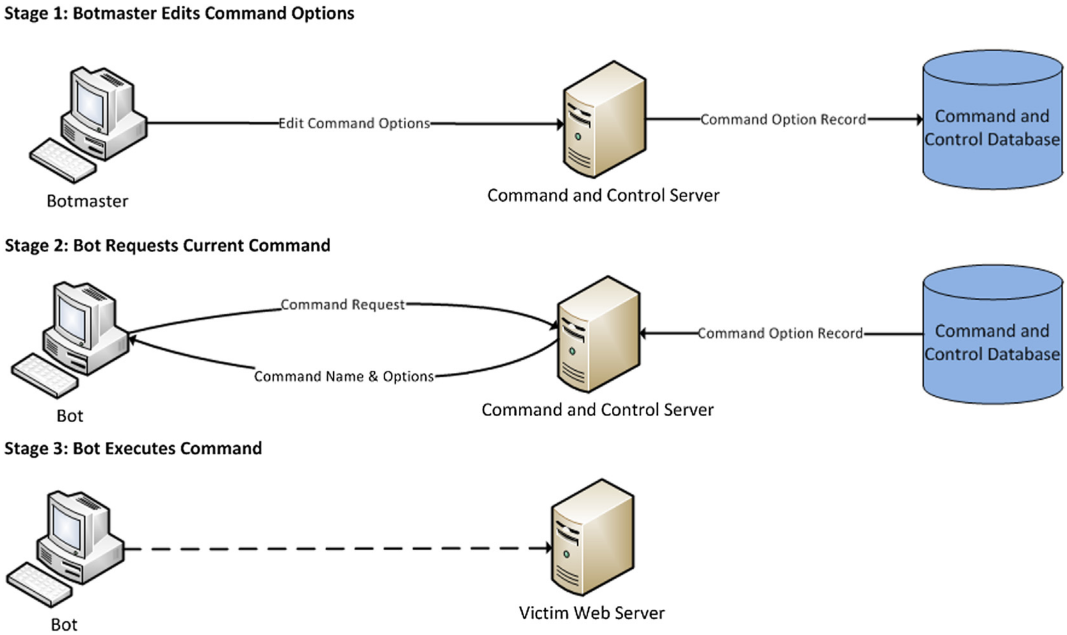

Stage 1: Botmaster Edits Command Options
Edit Command Options
Command Option Record
Command and Control Database
Botmaster
Command and Control Server
Stage 2: Bot Requests Current Command
Command Request
Command Name & Options
Command Option Record
Command and Control Database
Bot
Command and Control Server
Stage 3: Bot Executes Command
Bot
Victim Web Server

Fig. 7 – BlackEnergy communication overview.

Table 8 – Breakdown of the (2902) false positives per protocol.

Number of false alerts	Protocol/Application
981	Broadcast messages
1213	DropBox LAN sync protocol
549	Router Info Protocol
92	SSDP protocol
4	SRVLOC (service location) protocol
63	DNS protocol

Unlike the BlackEnergy dataset, the Weasel dataset is a mix of malicious and normal traffic. While the above results indicate that our model is good at detecting novel botnet traffic, it faces some challenges with unseen normal traffic when the traffic behavior exhibits some similarity with malicious traffic. A common approach to address such challenge is to improve the detector environmental awareness either by incorporating knowledge related to the target network topology in the configuration of the detector or to train the detector (before full operation) with sample normal traffic collected from the target network, which is a natural approach and makes sense.

Another common approach (used, for instance, by BotMiner (Gu et al., 2008b)) is to filter out traffic whose destinations are known to be legitimate servers. Likewise, by analyzing the false positives, we found out that several of them could be prevented with adequate filtering; Table 8 provides a breakdown of the false positives with corresponding applications. All these alerts correspond to known legitimate destination addresses. This indicates that by extending our detector using a whitelist of trusted destinations combined with port filtering, we can reduce dramatically the number of false positives. However, such approach is limited in novel botnet detection by the fact that a (new) bot can potentially impersonate legitimate services. For example, a bot can infect an enterprise web server by installing a malicious web application, which will be running inside the web server and can easily pass a port analysis filtering technique.

As alternative, a balanced approach that requires training of our detector using sample normal traffic collected from the target network (to be monitored), and integrates port filtering using a whitelist of trusted destination addresses will lead to an effective detector and allow eliminating all the false positives identified in Table 8.

6. Concluding remarks

Botnet represent currently one of the most predominant threats on the Internet. Effective online botnet detection is challenging because of the complexity and ever changing technology underlying botnets. Besides detection accuracy, some of the most important requirements of online botnet detection, which are largely unaddressed by most existing detection schemes include the following: early detection, novelty detection, and adaptability. Early detection poses some challenges because processing the huge size of traffic occurring in a typical network environment is a computationally intensive undertaking. The need for novelty detection stems from the fact that the characteristics of the network traffic as well as bots behaviors tend to change over time creating a concept-drift problem which is not well understood by most existing detection schemes. Furthermore, the fact that the network traffic data volume and characteristics change rapidly poses adaptability challenges to the detector, which ideally, should not require extensive retraining and should be capable of performing its functionalities while changing or updating the selected feature set.

In this paper, we have shown through different sets of experiments that our proposed model addresses adequately two of the above challenges, namely, early detection and novelty detection. Our proposed model allows 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 early 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. Experimental evaluation under various settings shows that our detector is able to achieve a true positive rate of over 90% and a false positive rate under 5%. We have also demonstrated that the proposed detector is capable of identifying new threats without training on existing malicious datasets, by testing sample novel botnets. This approach has several benefits over traditional approaches based on group behavior and anomaly detection. First, it does not require significant malicious activity to occur before detection as it can recognize command and control signals, and it does not require the group behavior of several bots before it can be confident about making a decision. There are, however, several challenges which must be overcome to realize a full implementation of such a system on a large internal network. Purely software based detectors capturing packets over standard network interfaces will likely result in an unacceptable performance impact if the detectors are distributed over individual devices on the network. Additionally, there would be significant challenges in installing and running individual detectors on network devices on any networks with more than a few hundred nodes. An alternative approach to detection may be to have a dedicated piece of hardware to monitor traffic on the entire network. Such an approach would be more costly but likely to result in better performance and may be the only realistic way of detecting malicious traffic by large network providers such as Internet Service Providers (ISPs) or organizations with large internal networks. The detection framework proposed in this work may still be improved in significant ways. For example, it may be possible to add evolving features to the detector such that it automatically adjusts the time interval and classifier over time. As P2P protocols become more prevalent in everyday non-malicious usage, it will become increasingly important for the detector to be able to distinguish at a high accuracy between malicious and novel non-malicious P2P protocols through retraining. Such retraining is currently not supported by the algorithm, and therefore it is vulnerable to a high false positive response when faced with novel normal network traffic. The system is particular weak to large changes in network traffic behavior, such as when a particular machine is repurposed for another task (i.e.: a web-server becomes a p2p file downloader). Improvements to the algorithm which would allow for incremental adjustments to the classifier without a thorough retraining session would greatly aid in the maintainability of such a detector in a live environment. For our future work, we plan to address the above mentioned challenges in order to create a framework that is truly robust and scalable, and as such can be used effectively for detection in large network environment.

REFERENCES

Al-Duwairi B, Al-Ebbini L. BotDigger: a fuzzy inference system for botnet detection. In: Proceedings of the fifth international conference on Internet and applications and services, Barcelona, 9–15 May 2010 2010. p. 16–21.

Feily M, Shahrestani A, Ramadass S. A survey of botnet and botnet detection. In: Proceedings of the third international conference on emerging security information, systems and technologies. IEEE Computer Society; 2009. p. 268–73.

Gao Z, Lu G, Gu D. A novel P2P traffic identification scheme based on support vector machine fuzzy network. In: Proceedings of 2nd international workshop on knowledge discovery and data mining, Jan. 2009 2009. p. 909–12.

Giroire F, Chandrashekar J, Taft N, Schooler E, Papagiannaki D. Exploiting temporal persistence to detect covert botnet channels. In: Proceedings of the 12th international symposium on recent advances in intrusion detection (RAID'09). Springer-Verlag; 2009. p. 326–45.

Grizzard JB, Sharma V, Nunnery C, ByungHoon Kang B, Dagon D. Peer-to-peer botnets: overview and case study. In: Proceedings of the first workshop on hot topics in understanding botnet (HotBots'07), Cambridge, MA, April 2007. Berkeley: USENIX Association; 2007.

Gu G, Porras P, Yegneswaran V, Fong M, Lee W. BotHunter: detecting malware infection through IDS-driven dialog correlation. In: Proceedings of the 16th USENIX security symposium, Boston, MA, USA 2007. p. 167–82.

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.

Gu G, Perdisci R, Zhang J, Lee W. BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In: Proceedings of the 17th USENIX security symposium, San Jose, CA, USA 2008.

Holz T, Steiner M, Dahl F, Biersack E, Freilin F. 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. Berkeley: USENIX Association; 2008.

Lawrence Berkeley National Laboratory and ICSI. LBNL/ICSI enterprise tracing project. LBNL enterprise trace repository. [Online], http://www.icir.org/enterprise-tracing; 2005.

Leonard J, Xu S, Sandhu R. A framework for understanding botnets. In: Proceedings of international conference on availability, reliability and security. IEEE Computer Society; 2009. p. 917–22.

Li J, Zhang S, Liu S, Xuan Y. P2P traffic identification technique. In: Proceedings of the international conference on computational intelligence and security, Harbin, China, 15–19 Dec. 2007 2007. p. 37–41.

Livadas C, Walsh R, Lapsley D, Strayer WT. Using machine learning techniques to identify botnet traffic. In: Proceedings of 2nd IEEE LCN workshop on network security 2006. p. 967–74.

Masud M, Gao J, Khan L, Han J, Thuraisingham B. Mining concept-drifting data stream to detect peer-to-peer botnet traffic. Univ. of Texas at Dallas Tech. Report UTDCS-05-08, http://www.utdallas.edu/mmm058000/reports/UTDCS-05-08.pdf; 2008.

Nazario J. BlackEnergy DDoS bot analysis. Technical Report. Arbor Networks. p. 11. Available at, http://atlas-public.ec2.arbor.net/docs/BlackEnergy+DDoS+Bot+Analysis.pdf; October 2007.

openpacket.org. Zeus/Zbot sample traffic and C& C traffic. Retrieved October 29, 2012, from www.openpacket.org.

Rajab MA, Zarfoss J, Monrose F, Terzis A. A multifaceted approach to understanding the botnet phenomenon. In: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement. New York: ACM; 2006.

Saad S, Traore I, Ghorbani A, Sayed B, Zhao D, Lu W, et al. Detecting P2P botnets through network behavior analysis and machine learning. In: Proceedings of 9th annual conference on privacy, security and trust (PST2011), July 19–21, 2011, Montreal, Canada 2011.

Shiravi A, Shiravi H, Tavallaee M, Ghorbani A. Toward developing a systematic approach to generate benchmark datasets for intrusion detection. Computers & Security, Elsevier 2012;31(3): 357–74.

Sinclair G, Nunnery C, Kang BB. The Waledac protocol: the how and why. In: Proceeding of 4th IEEE international conference on malicious and unwanted software 2009. p. 69–77.

Snort Roesch M. Lightweight intrusion detection for networks. In: Proceedings of the 13th USENIX conference on system administration 1999. p. 229–38.

Sperotto A, Schaffrath G, Sadre R, Morariu C, Pras A, Stiller B. An overview of IP flow-based intrusion detection. IEEE Communications Surveys & Tutorial 2010;12(3):343–56.

Szab G, Orincsay D, Malomsoky S, Szab I. On the validation of traffic classification algorithms. In: Proceedings of the 9th international conference on passive and active network measurement. Berlin, Heidelberg: Springer-Verlag; 2008. p. 72–81.

The Honeynet Project. French Chapter [Online] http://www.honeynet.org/chapters/france.

Villamarn-Salomn R, Brustoloni JC. Bayesian bot detection based on DNS traffic similarity. In: Proceedings of the 2009 ACM symposium on applied computing, Honolulu, Hawaii 2009. p. 2035–41.

Wang P, Sparks S, Zou CC. An advanced hybrid peer-to-peer botnet. In: Proceedings of the first workshop on hot topics in understanding botnets. Berkeley, CA, USA: USENIX Association; 2007.

Wang B, Li Z, Tu H, MaWang J. Measuring peer-to-peer botnets using control flow stability. In: Proceedings of the fourth international conference on availability, reliability and security, March 16–19, 2009, Fukuoka, Japan 2009. p. 663–9.

Witten IH, Frank E, Trigg L, Hall M, Holmes G, Cunningham SJ. Weka: practical machine learning tools and techniques (Working paper 99/11). Hamilton, New Zealand: University of Waikato, Department of Computer Science; 1999.

Wurzinger P, Bilge L, Holz T, Goebel J, Kruegel C, Kirda E. Automatically generating models for botnet detection. In: Proceedings of the 14th European conference on research in computer security (ESORICS 2009). Lecture Notes in Computer Science, vol. 5789. Springer Verlag; 2009. p. 232–49.

Yu X, Dong X, Yu G, Qin Y, Yue D, Zhao Y. Online botnet detection based on incremental discrete Fourier transform. Journal of Networks 2010;5(5).

Zeidanloo HR, Rouhani S. Botnet detection by monitoring common network behaviors. Lambert Academic Publishing, ISBN 9783848404759; March 2012.

Zhao D, Traore I, Ghorbani A, Sayed B, Saad S, Lu W. Peer-to-peer botnet detection based on flow intervals. In: Proceedings of IFIP international information security and privacy conference (SEC 2012), 4–6 June 2012, Crete, Greece 2012.

David Zhao obtained a Bachelor of Software Engineering, and a M.A.Sc. degree from the Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, Canada. He is currently working as Software Engineer, Yupiq Solutions Ltd., Victoria, BC, Canada. His research interests include malware detection, biometric security, and machine learning.

Issa Traore is the CEO and co-founder of Plurilock Security Solutions Inc. (www.plurilock.com) He has been with the faculty of the Electrical and Computer Engineering Department of the University of Victoria since 1999, where he is currently an Associate Professor. Dr. Traore is also the founder and Director of the Information Security and Object Technology (ISOT) Lab (www.isot.ece.uvic.ca). He obtained in 1998 a PhD in Software Engineering from the Institute Nationale Polytechnique of Toulouse, France. His main research interests are biometrics technologies, intrusion detection systems, and software security.

Bassam Sayed received the B.Sc. degree in computer science from Helwan University, Cairo, Egypt, and the M.A.Sc. degree from the Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC, Canada, where he is currently pursuing the Ph.D. degree. His current research interests include behavioral biometrics, Botnet detection, and web-based malware detection.

Wei Lu received the Ph.D. degree in Electrical and Computer Engineering from the University of Victoria in 2005. He was a Postdoctoral Researcher with the German Research Center for Artificial Intelligence and worked with Q1 Labs Inc. (Security Systems Division, IBM since October 2011) as a Secure Software Engineer. He joined University System of New Hampshire/Keene State College as an Assistant Professor of Computer Science in 2010 and his general areas of research interests include Internet traffic classification, intrusion detection and prevention, and the application of artificial intelligence and machine learning techniques on the practical computer and network security issues.

Sherif Saad received his B.Sc. in Computer Science from Helwan University, Egypt (2003), M.Sc in Computer Science from Arab Academy for Science, Technology and Maritime Transport, Egypt (2007). In 2008 he received the University of Victoria Fellowship and started his PhD in the Department of Electrical and Computer Engineering of the University of Victoria. His primary research interests are in advancing machine-learning methods and their application to computer and network security. Since 2009, he is working as an information security engineer for Plurilock Security Solutions (www.plurilock.com/).

Ali A. Ghorbani received the B.Sc. degree from the University of Tehran, Iran, in 1976, the M.Sc. degree from George Washington University, Washington, DC, in 1979, and the Ph.D. degree from the University of New Brunswick (UNB), Fredericton, NB, Canada, in 1995. He is currently a Professor and Dean with the UNB, where he is the Director of Information Security Center of Excellence. He is the Coeditor-in-Chief of the Computational Intelligence: An International Journal, and an Associate Editor of the International Journal of Information Technology and Web Engineering. His current research interests include web intelligence, network security, complex adaptive systems.

Dan Garant is a senior undergraduate at Keene State College majoring in Computer Science and Mathematics. He works as a software engineering intern at Vision Financial and a research assistant at University of Massachusetts at Amherst. He participated in the 2012 REU program at UMass-Amherst, focusing on organizations in multi-agent systems. His research interests lie in knowledge discovery, data mining, machine learning, and applications of these fields in network security.