Botnet Detection Based on Network Behavior

W. Timothy Strayer[1], David Lapsely[1], Robert Walsh[1], and Carl Livadas[2]

[1] BBN Technologies, Cambridge, MA 02138
strayer|dlapsely|rwalsh@bbn.com
[2] Intel Research, Santa Clara, CA 95054
carlx.livadas@intel.com

Current techniques for detecting botnets examine traffic content for IRC commands, monitor DNS for strange usage, or set up honeynets to capture live bots. Our botnet detection approach is to examine *flow characteristics* such as bandwidth, packet timing, and burst duration for evidence of botnet command and control activity. We have constructed an architecture that first eliminates traffic that is unlikely to be a part of a botnet, classifies the remaining traffic into a group that is likely to be part of a botnet, then correlates the likely traffic to find common communications patterns that would suggest the activity of a botnet. Our results show that botnet evidence can be extracted from a traffic trace containing over 1.3 million flows.

1 Introduction

Botnets are one of the most dangerous species of network-based attack today because they involve the use of very large, coordinated groups of hosts for both brute-force and subtle attacks. These large groups of hosts are assembled by turning vulnerable hosts into so-called *zombies*, or *bots*, after which they can be controlled from afar. A collection of bots, when controlled by a single command and control (C2) infrastructure, form what is called a *botnet*. Botnets obfuscate the attacking host by providing a level of indirection — the attack host is separated from its victim by the layer of zombie hosts, and the attack itself is separated from the assembly of the botnet by an arbitrary amount of time.

Botnets derive their power by scale, both in their cumulative bandwidth and in their reach. Botnets can cause severe network disruptions through massive distributed denial-of-service attacks, and the threat of this disruption can cost enterprises large sums in extortion fees. They are responsible for a vast majority of the spam on the Internet today. Botnets are also used to harvest personal, corporate, or government sensitive information for sale on a thriving organized crime market. They are a reusable and renewable resource.

Governments are taking the threat of botnets seriously. In August 2005, Britain's NISCC (National Infrastructure Security Coordination Centre, the UK equivalent

to US-CERT) issued a warning about the increase in trojan activity targeting UK government networks, stating that "the attacker's aim appears to be covert gathering and transmitting of commercially or economically valuable information" [22]. In November 2005, the discovery of a botnet in US Department of Defense [32] caused the head of DoD networks to issue an "information assurance standdown," followed by a full sweep of all DoD networks [5].

Efforts are underway to quantify the botnet problem, detect the presence of botnets, and design defenses against attacks by botnets. In academia, for example, Ramachandran *et al.* have been studying the effectiveness of monitoring queries to DNS blackhole lists to find bot masters looking to see if their bots have been blacklisted [23]. Dagon *et al.* use diurnal models to compare the propagation rate for different botnets [4]. Karasaridis *et al.* use suspicious host activity reports (scanning ports, emailing spam and virus, generating DDoS traffic) as indicators of flows to analyze [14]. And Kandula *et al.* suggest ways for websites and other services to thwart bot and other mechanical agents by using Turing tests [13].

Non-profit and volunteer organizations are involved. The Honeynet Project [31], for example, has done extensive work on capturing live bots and characterizing botnet activities, and a group of white-hat vigilantes is scouring the Internet looking for evidence of botnets [21]. Industry and federally funded centers are also active: Symantec publishes a semi-annual *Internet Security Threat Report* [30] identifying trends in attack mechanisms, and CERT maintains a Vulnerability Notes Database [1] with information on botnet and other attack vectors.

Determining the source of a botnet-based attack is a particular challenge. First, there is a distinction between the attack and the attack mechanism. For single-flow [26] and "stepping stone" chained-flow [37] attacks, the flow is both the mechanism and the attack, but for botnets, the mechanism (the botnet) is constructed and maintained independently of how it is used. Second, there is a difference in what constitutes the "attack origin." Tracing flow-based attacks attempts to yield a single responsible host; with botnets, every zombie host is an attacker. Finally, most flow-based traceback systems adopt a reactive approach to attacks; the tracing of packets back to their origin hosts is triggered after an attack is detected. Botnets can exist in a benign state for an arbitrary amount of time before they are used for a specific attack, affording some opportunity to identify them prior to the attack.

We are interested in botnets with tight command and control infrastructures, as shown in Figure 1. IRC is the most common botnet C2 mechanism [10, 11, 16, 18, 19, 31] because it is scalable and easy to hide within. While instances of botnets with looser control structures, such as those that use peer-to-peer networks, are increasing, IRC-style C2 is still the most prevalent because it is scalable and provides instantaneous control over the bots.

In botnets that use the chat style of command and control, the attacker issues commands to the zombie hosts via a "rendezvous point," which is usually an IRC server. The rendezvous point may or may not be a compromised machine — there are many public IRC servers that host unmonitored channels. The attacker and the zombie hosts subscribe to the same IRC channel. The attacker issues commands and the bots respond through that channel.

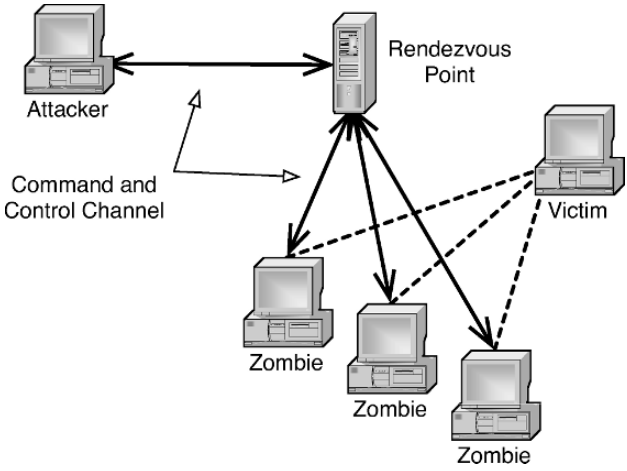

Fig. 1. Actors in IRC-Based Botnet Architecture

This chapter presents a system for detecting the presence of a botnet and identifying the rendezvous point using passive traffic analysis. (Some initial results were presented in [29].) Our goal is to determine if we can find evidence of botnet activity by only monitoring network traffic, and not by examining the traffic content, relying on port numbers (IRC's is 6667), or by watching DNS servers. We adopt a proactive approach by identifying hosts that are likely part of a botnet *before an attack* by extracting and analyzing flow characteristics that seem to match botnet C2 traffic.

Our technique employs a pipeline of increasingly more complex analyzers, filtering out unlikely flows along each step, so that the most computationally intensive analysis is done on a dramatically reduced traffic set. First, individual flows are subjected to a series of filters and classifiers to eliminate as many of the flows as possible, while being somewhat conservative so that botnet flows are not likely to be eliminated. Next, the flows are correlated with each other, looking for groups of flows that may be related by being part of the same botnet. Finally, the topological information in the correlated flows is examined for the presence of a common communication hub.

2 Approach

Since the vast majority of botnets are controlled using variations on IRC bots, many botnet detection systems begin by simply looking for chat sessions (TCP port 6667) [12], and then examining the content for botnet commands [2]. Like many client-server protocols, however, the use of a standard port number is largely just a suggestion. Also, relying on having access to the packet contents and, even with that access, being able to identify botnet commands, is an overly simplistic assumption.

Our system assumes only that the botnet command and control (C2) infrastructure is based loosely on IRC.

2.1 Characterization of IRC-based C2 Flows

IRC-based botnets currently dominate as the preferred deployment technique. This reflects the freely available bot-building source code, allowing attackers to focus on botnet applications rather than on architecting and coding "mere plumbing." IRC is implemented through text-based interactions. Strings are sent to the chat server, which replicates that data to each client. In the case of botnets, the clients are zombies, and botnet commands are special strings.

We use chat traffic as an initial proxy for botnet C2 traffic. By looking at example botnet commands [31], the important insight is that C2 messages are brief in addition to being text-based. In the absence of access to extensive botnet traces, we characterize chat flows to identify how we can separate the C2 channel from other Internet traffic.

Specifically, there are four notable points. First, identification of chat is a statistical problem. For each attribute of a flow, chat flows are spread across the spectrum of values. Instead of a deterministic decision, one is left with a probabilistic conclusion, complete with the risk of false positives and false negatives.

Second, identification of chat in the absence of well-known ports and access to the packet content is a difficult problem. Flows can be winnowed into likely chat and likely non-chat classifications, but the likely chat classification will certainly include a number of non-chat flows.

Third, consideration of attributes in isolation is a good start, but is not sufficient — it is equivalent to using independent probabilities to evaluate the traffic. Stronger techniques based upon interdependent conditional probabilities may be needed as well.

Finally, the resulting characterization is good for guiding the construction of efficient filters for data reduction. By reducing the data set, even if it contains some false positives, later steps can take advantage of more computationally intensive approaches.

2.2 The Processing Pipeline

Figure 2 shows our traffic-processing pipeline. Packet traces (in our case these are recorded traces, but there is no reason the input cannot be live) are fed into a series of quick reduction filters. With some *a priori* knowledge, one can also imagine a set of white lists and black lists based on known good sites (packets to or from eBay, for example, are very unlikely to be part of a botnet) and bad sites (those places on a watch list, for example). Other filters examine simple flow attributions such as duration or average packet size.

After the initial filters, the remaining flows are passed through a flow classification engine based on machine learning techniques. The classifiers attempt to group

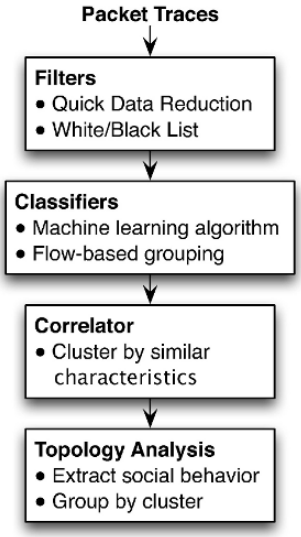

Fig. 2. Botnet Detection Processing Pipeline

flows into broadly defined categories. Those flows that appear to have chat-like characteristics are passed on to the correlator stage.

The correlator does a pairwise examination of the remaining flows looking for flows that are behaving in a similar manner, as one might expect of two flows generated by the same application. Botnets are so large that commands are issued to the whole group, or large subgroups, and not to individuals. Flows that are correlated are passed on to topological analysis, where "social topology" is applied to determine which flows share a common controller.

The result of this pipeline is a (hopefully) small set of flows that show a fair amount of evidence that they are related and are part of a botnet. The pipeline does not prove the flows are part of a botnet; rather, the flows that survive strongly suggest closer examination. This examination may be deep, if there is access to the hosts that are the flow endpoints, as may happen in an enterprise or campus, or the examination may be limited to listing the flows and the flow endpoints in a watch list for later use if a botnet-based attack occurs. Knowing the social structure of a group of hosts prior to an attack is better than trying to piece the structure together during the attack.

2.3 Source of Background Traffic

It would be too contrived to try to create a large dataset of both background and botnet traffic using a tightly controlled testbed. Instead, we incorporated a background traffic data set recorded from true Internet use. We chose packet traces collected on the Dartmouth campus under their CRAWDAD project [15]. The traces are a complete set of TCP/IP headers from the campus wireless, taken over a period of four

months (November 1, 2003 to February 28, 2004) from a variety of campus locations. No payloads were included in the trace.

In all, the traces were 164 GBytes compressed, and approximately 3.8 times that amount when uncompressed. This large trace set means that we truly are looking for the needle (botnet C2 flows) in a haystack.

From this set of traces, we selected a subset of traces that corresponded to a particular building that we shall label "Building X." We believed the traces from Building X to be representative of "typical" Internet background traffic for our botnet scenario. We then selected a reference time point of Monday November 10, 14:30 EST, 2003 as the time at which we would attempt to detect our synthesized botnet (the needle) in the presence of this background traffic (the haystack). Our detection process examined all of the uni-directional flows of data between hosts from the start of the Building X traces on Monday November 1, 2003 at 23:12 EST until just after our reference time point on Monday November 10, 2003 at 14:30 EST. In total, 1.34 million uni-directional data flows were examined.

2.4 Source of Botnet Traces

In order to generate traffic that was representative of real botnet traffic, we implemented a benign bot based on the "Kaiten" bot, a widespread bot that has readily downloadable source code. The Kaiten bot was implemented in C using approximately 1000 lines of code. We reverse engineered the Kaiten code and then reimplemented it.

The original Kaiten bot had a repertoire of TCP- and UDP-based attacks. Our bot implementation does not implement these attacks. Like the Kaiten bot, our bot provides a number of remotely controlled features, including a mechanism to execute arbitrary commands on the bot client, HTTP download capability, a flexible multi-process architecture, a highly configurable architecture and a rich command set.

In order to obtain traces of actual botnet traffic, we constructed a botnet testbed within BBN's production network. Our setup consisted of an IRC server (rendezvous point), a code server, 10 zombie hosts, and an attacker. Figure 3 shows the topology of our botnet testbed. The attacker, the rendezvous point, and one zombie host reside on an external network. Nine zombies and the victim were hosted within the BBN network. The code server was a large well known public Internet site.

We used this test facility to obtain actual traces of the communications between the various botnet entities while the botnet was in operation. Our experiments entailed using the IRC server to instruct the zombies to download attack code from the code server and to subsequently launch a coordinated TCP "attack" on the victim host. The traces collected involved *ssh* transmissions used for setting up and monitoring the experiments, IRC traffic between the bots and the IRC server, *http* traffic between the zombies and the code server (for downloading the attack code), and the TCP traffic involved in the coordinated TCP attack on the victim host. The setup and the launch of the attack were successively repeated in order to increase the amount of trace data collected.

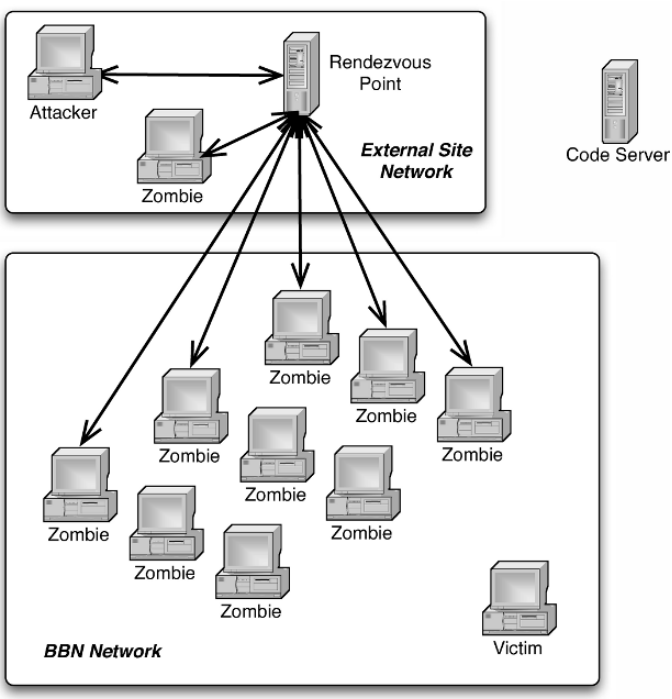

Fig. 3. Botnet Trace Collection Testbed

We collected 539 flows associated with our botnet using tcpdump at the IRC server. Forty two of these flows were C2 flows. We merged this botnet trace with the Dartmouth traffic data set in order to create a test data set that contained ground truth that could be verified after all of the data reduction filters and other analyzers have been applied. Our botnet was active on the order of hours, while the Dartmouth traces span four months, exacerbating the vast size difference between the needle and the haystack.

3 Filtering Stage

We recognize that the statistical nature of the problem creates a trade-off between keeping as many botnet C2 flows as possible and reduction of the data set to the meaningful subset of flows to speed later steps. The selection of the cutoff for quick filtering for data reduction requires both quantitative statistical information and human judgment. Even if the selection of the cutoff were phrased in terms of meeting a false positive or a false negative goal, that goal is based upon judgment. The filters and filter parameters we chose reflect this.

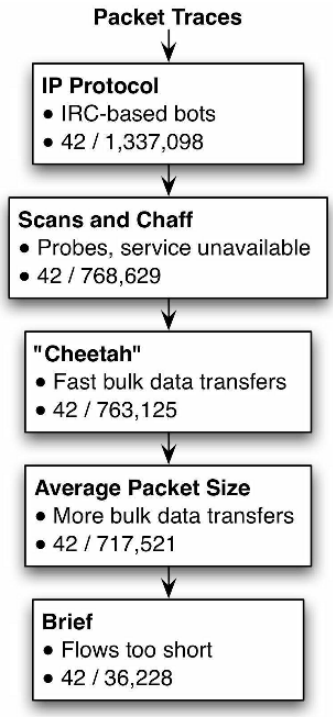

Fig. 4. Filtering Out Flows Not Likely Part of a Botnet

There were five distinct filters in this stage, as shown in Figure 4. The first filtered by IP protocol to select TCP-based flows, resulting in 1,337,098 flows. Since the bot was derived from an IRC-style TCP base, all of the ground-truth botnet C2 flows were TCP based. All of the C2 flows survived this filter.

The second filter removed the nuisance port-scanning chaff, reducing the data set to 786,629 flows. Flows containing only TCP packets with SYN or RST flags indicate that communication was never established, and so provide no information about chat or botnet C2 flows. No application-level data was transferred by these flows. Unfortunately for today's Internet, probes of system vulnerabilities are commonplace. While SYN-RST exchanges indicate suspicious activity that may be worth investigation, they do not assist with characterizing botnet C2 flows. About 43% of the flows are eliminated by this step. Again, all of the ground-truth botnet C2 flows survived the filter.

Since botnets do not sustain bulk data transfers, the next filter removed high bitrate flows. Peer-to-peer file sharing is a significant load on the Internet, and may take place on chat ports by coincidence (since the chat port is not reserved) or by intent (to avoid identification and filtering). Dropping bulk transfers (flow bandwidth greater than 8 Kb/s with at least 50 packets) also eliminates software updates and rich web page transfers. Yet, filtering the high bit-rate flows had a small effect. About

1% of the flows are dropped, leaving 763,125. From a flow perspective, this is a minor amount, but from a packet and forensic archive perspective this represents a worthwhile effort. Again, all of the bot C2 flows survived the filter.

Chat (and botnet C2 commands) generally generate small packets. Using a 300-byte packet size cutoff for the chat packets in the Dartmouth data set shows that about 0.25% of the chat traffic would be falsely rejected and 72% of the non-chat flows are eliminated. Since there are several orders of magnitude more non-chat flows than chat flows, filtering exclusively on average packet size would cut the amount of data to process in half; since this filter comes fourth, it has a relatively moderate effect. About 6% of the flows are dropped, leaving 717,521. All of the ground-truth botnet C2 flows survived the filter.

The fifth filter drops brief flows (less than 2 packets or 60 seconds) from consideration. Real chats and botnets are likely not well represented by excessively short duration flows. This filter has a significant effect, reducing the data by a factor of about 20, dominating even the elimination of the port-scanning activities. All of the ground-truth botnet C2 flows survived the filter.

Overall, the data set is reduced by a factor of about 37, from 1,337,098 TCP flows down to 36,228, while still preserving the ground-truth botnet C2 flows. This filtering stage avoided the use of TCP port numbers, and therefore is relevant to situations where applications may be masquerading on unexpected ports. Furthermore, this significant data reduction resulted without the use of white-listing services as trusted IP address and port number combinations.

4 Classifier Stage

Once the simple filters have reduced the data set, the next step is to process the data set using more sophisticated flow classification techniques. Several techniques have been developed to automatically identify (and often classify) various types of communication streams. Some use clues from the traffic content. Dewes *et al.* [6], for instance, proposed a scheme for identifying chat traffic that relies on a combination of discriminating criteria, including service port number, packet size distribution, and packet content. Sen *et al.* [25] used a signature-based scheme to discern traffic produced by several well-known P2P applications by identifying particular characteristics in the syntax of packet contents exchanged as part of the operation of the particular P2P applications.

Other flow classification approaches focus on the use of statistical techniques to characterize and classify traffic streams. Roughan *et al.* [24] used traffic classification for the purpose of identifying four major classes of service: interactive, bulk data transfer, streaming, and transactional. They investigated the effectiveness of using packet size and flow duration characteristics, and simple classification schemes were observed to produce very accurate traffic flow classification.

In a similar approach, Moore and Zuev [20] applied variants of the Naïve Bayesian classification scheme to classify flows into 10 distinct application groups. The authors also searched through the various traffic characteristics to identify those

that are most effective at discriminating among the various traffic flow classes. By also identifying highly correlated traffic flow characteristics, this search was also effective in pruning the number of traffic flow characteristics used to discriminate among traffic flows. Highly correlated characteristics provide comparable and, often, redundant information about the traffic flows. Thus, in many cases it suffices to use only one of the correlated characteristics to discriminate among traffic flows.

Since IRC-type botnet C2 flows share many characteristics with normal IRC chat flows, we adopt and build upon the above statistical flow classification techniques to discriminate among IRC and non-IRC traffic (see Livadas *et al.* [17]). The focus on IRC traffic simplifies the training step because the default IRC port (namely, port 6667) can be used to accurately identify and label IRC traffic for training and ground truth.

We considered three machine learning classification algorithms, namely J48 decision trees (the WEKA [34] implementation of C4.5 decision trees [8]), Naïve Bayes, and Bayesian Networks, and evaluated the performance of each classifier using the false negative rate (FNR) and the false positive rate (FPR). The relative importance of each of these metrics depends on the ultimate use of the classification results. A low FNR attempts to minimize the fraction of the IRC flows will be discarded, while a low FPR attempts to minimize the amount of non-IRC flows included. We explored the effectiveness of these machine learning techniques along three dimensions: (1) the subset of characteristics/features used to describe the flows, (2) the classification scheme, and (3) the size of the training set size.

Table 1 summarizes the flow characteristics that we collected for each of the flows in the Dartmouth traces. The characteristics in the top of the table were not used for classification purposes — they either involve characteristics that seemed inconsequential in classifying flows, or are accumulated quantities, which are indirectly captured by the corresponding rates or percentages and the flow duration. Our experiments revealed that the following attributes have high discriminatory value: duration, role, average bytes per packet (Bpp), average bits per second (bps), and average packets per second (pps). Among these, the Bpp provided the most discriminatory power.

Figure 5 depicts the FNR vs. FPR scatter plot for several runs of J48, Naïve Bayes, and Bayesian Networks for the labeled Building X trace. Each data point corresponds to a different subset of the initial flow attribute set. The figure reveals clustering in the performance of each of three classification techniques. Naïve Bayes seems to have low FNR, but higher FPR. The Bayesian Networks technique seems to have low FPR, but higher FNR. J48 seems to strike a balance between FNR and FPR.

Only the Naïve Bayes classifiers were successful in achieving low FNR in the case of our botnet testbed IRC flows — notably, one of our Naïve Bayes classifiers accurately classified 41 out of the 42 botnet testbed IRC flows, thus achieving an FNR of 2.17%. In contrast, the J48 and the Bayesian Networks classifiers, possibly tuned too tightly to the training set, performed very poorly with FNRs of 28.26 and 19.57% respectively. However, while the Naïve Bayes classifiers had a low FNR, they also had a high FPR of 30.41%. Of the 36,136 non-botnet flows, 11,004 were

Table 1. Traffic Flow Characteristics

start/end	Flow start/end times
IP-proto	IP protocol of flow
TCP flags	Summary of TCP SYN/FIN/ACK flags
pkts	Total pkts exchanged in flow
Bytes	Total Bytes exchanged in flow
pushed pkts	Total packets pushed in flow
duration	Flow duration
maxwin	Maximum initial congestion window
role	Whether client or server initiated flow
Bpp	Average Bytes-per-packet for flow
bps	Average bits-per-second for flow
pps	Average packets-per-second for flow
PctPktsPushed	Percentage of packets pushed in flow
PctBppHistBin0–7	Percent of packets in one of eight packet size bins; these variables collectively form a histogram of packet size for flow
varIAT	Variance of packet inter-arrival time for flow
varBpp	Variance of Bytes-per-packet for flow

classified as belonging to the botnet. After training on the flows yielded from the earlier heuristic filtering stage, our best-performing classifiers achieved a 70% reduction in the number of candidate chat flows. Presuming that such performance would be routinely achievable in this stage, the 36K flows yielded from the heuristic filtering stage would be further reduced to 11K flows. In the case of the testbed flows, our best-performing classifiers retained 41 of the 42 chat flows.

Despite their promise, the training and performance of classifiers was quite sensitive to the flow attributes used, the training set, and the number of flows used for training. Thus, prior to their use in a deployable system we expect that further effort would be needed in order to identify the most beneficial flow characteristics and training set. For the processing of our testbed experiment, we bypassed the classification stage and proceeded directly from filtering to correlation.

5 Correlation Stage

The filters and classifiers have reduced the traffic data set from almost 1.34 million flows to about 36 thousand, but recall that these flows span a four-month period. Our next stage, correlation, looks for relationships between two or more flows that suggest that they are part of the same botnet. The question about whether one flow

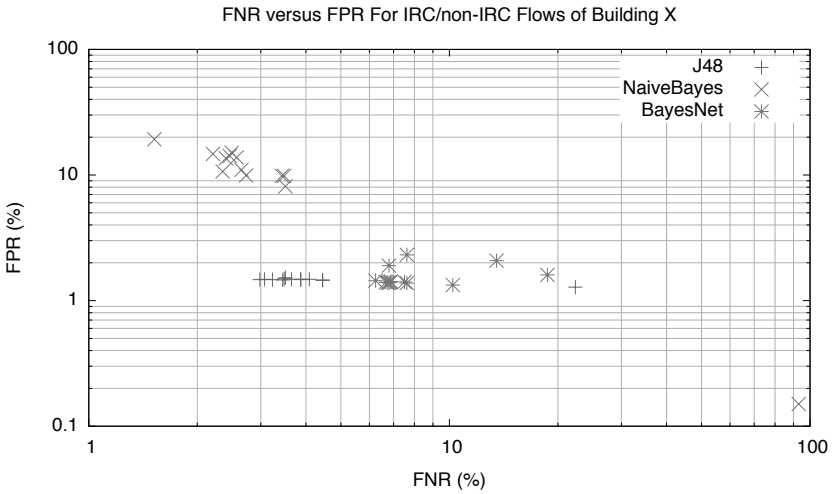

Fig. 5. FNR and FPR of J48, Naïve Bayes, and Bayesian Net Classification Schemes for IRC/non-IRC Flows of Building X

is correlated with another only makes sense if the two flows are active at the same time, so while we have four months of data, the correlation stage is run at a particular instance in time. The question is: Which flows are correlated at this moment?

We picked a time during the data when we knew the botnet was active. There were 95 post-filtered flows active at that time, where 20 of these flows were the ground-truth botnet C2 flows (a forward and a reverse flow from each of the 10 zombie hosts to the rendezvous point).

5.1 Flow Correlation

Two flows are said to be *correlated* when they exhibit one or more common properties. In general, there are three reasons that two flows exhibit common properties:

- They are the product of similar applications, such as those applications that transfer bulk data as quickly as possible
- There is a causal relationship, such as in remote logins or proxies, where an event on one flow causes an event to occur on another flow
- There is one transmitter and multiple receivers, such as in multicast, where one message is transmitted to many receivers

The first reason is a product of the nature of network protocols. TCP behaves the same no matter what application is driving it. If two applications present large files for transfer, there is little at the packet level to distinguish the traffic outside of the addressing information.

The second correlation reason speaks to the so-called stepping stone detection problem, where an attacker remotely logs into one host, then from there remotely

logs into another host, repeating to form a chain of remote logins. The attacker sees the login shell of the last host, and anything typed in at the local keyboard cascades its way to the pseudo terminal at the last host. The cascading of the data is what provides the casual relationship among the flows in the chain.

The third reason for correlation happens because the same data is being sent to different receivers, so naturally the set of flows will show similar characteristics. Botnets that use IRC for the command and control channel essentially form multicast groups via a series of operations on unicast connections.

No matter the reason for correlation, any algorithm that sets out to determine which pairs of flow are correlated must begin with this question: What is a sufficient description scheme for flows so that the algorithm can determine if two flows are correlated under a particular meaning of correlation?

Flow Description

A flow is defined as a set of packets that belong to the same instance of communication between an application at a source host, and an application at a destination host. The most common way to identify a particular TCP or UDP flow is using a 5-tuple of values from the packets' layer 3 and 4 headers: the source and destination IP addresses, the source and destination port numbers, and the protocol identifier number. These five values definitively identify a particular instance of communication between a source host application and destination host application.

It is one thing to uniquely identify the flow; it is something all together different to uniquely describe a flow. Describing an object allows that object to be compared and contrasted with other objects. The same is true for flows. Choosing a certain set of characteristics and quantizing those characteristics provides one means of capturing describable aspects of the flow for comparison with other flows.

Certainly a flow can be completely described using a full packet trace, as one might get from a tool such as tcpdump. Such a trace lists when each packet event occurred, what was inside the packet's header, and what data each packet was carrying. Since a flow can be arbitrarily long, a packet trace can be arbitrarily long.

Packet trace files are a complete description, but they are not a compact one. It may be sufficient to extract and efficiently express a set of flow characteristics as a proxy for the full flow description.

Flow Characteristics

Flow characteristics fall into two categories: static characteristics that do not change over the lifetime of the flow, and dynamic characteristics that vary as the flow progresses through time. The immutable information kept in the IP and TCP/UDP headers of a packet is a good source of static characteristics. These include the values that form the flow identification 5-tuple — source and destination IP address, source and destination port numbers, and protocol. Flow start and stop times, and the flow's duration, are examples of static characteristics that are not carried in the packet.

Dynamic characteristics can also be drawn from the packet header and payload information, such as packet size values, flow control window settings, IPid values, protocol flag settings, and application data. Looking outside of the packet, dynamic characteristics include packet arrival and departure times. Further dynamic characteristics can be derived, such as throughput (amount of data transferred divided by the transfer duration), and burst times (groupings of packet arrivals or departures that are close in time).

Among the common dynamic flow characteristics that are easily expressed as a time series are:

- Packet event times
- Packet inter-arrival times
- Inter-burst times
- Bytes per packet
- Cumulative bytes per packet
- Bytes per burst
- Periodic throughput samples

Flow Correlation Algorithms

The most common flow correlation algorithms compare connections to see if they might be stepping stones — the causal relationship noted above. Our aim is to find correlations between flows based on a multicast relationship. We hypothesize that stepping stone correlation algorithms can be used to find botnets. Consequently, we will take a quick survey of stepping stone correlation algorithms looking for one that may be appropriate for our purposes.

Since traffic is often encrypted, flow correlation algorithms usually compare connections based on some characteristic other than packet content. Most correlation algorithms use only a single characteristic to describe packet flows. For example, an algorithm might describe a flow based on its packet inter-arrival times. Whatever the characteristic may be, it is chosen so that it can be used to identify related connections. These algorithms use the characteristic values as inputs into one or more functions that compare flows. The comparison function(s) create a metric used to decide if the flows are correlated. If the correlation between two flows is strong enough, one might decide that the flows are a stepping stone pair. Often, this decision is made by comparing the metric to a threshold.

Zhang and Paxon [37] describe a stepping stone detection method based on comparing the end times of "off periods," or idle times, in two data streams. The characteristics they focus on is the timing of the edge of bursts. Yoda and Etoh [35] describe an algorithm based on the difference between the average propagation delay and the minimum propagation delay between the two connections. Their flow characteristic is the round-trip time. Wang *et al.* [33] present a stepping stone identification scheme that uses similarity function over a vector of inter-packet delay measures (their flow characteristic) between two packet streams.

The aim of some approaches is to assert guaranteed false positive and negative rates under delay and chaff perturbations. Blum *et al.* [3] designed a stepping stone detection algorithm based on the deviation in the number of packets in each connection. Zhang *et al.* [36] propose three schemes that match packets from one flow to packets in a second flow to detect stepping stone connections. Both Blum and Zhang use packet counts as the flow characteristic. He and Tong [9] propose four packet counting (their flow characteristic) strategies — two algorithms based on bounded memory or bounded delay perturbation and chaff, and two algorithms that handle timing perturbation and chaff insertion simultaneously.

Strayer *et al.* [28] proposed a correlation algorithm that examines the causal relationship between packet events based on the assumption that, because networks attempt to operate efficiently, the likelihood of a transmission on one connection being a response to a prior receipt on another generally decreases as the elapsed time between them increases. Packet arrival time is the flow characteristic maintained here.

Donoho *et al.* [7] use character counts at different time scales, along with an assumption that there is a "maximum delay tolerance" to produce theoretical limits on the ability of attackers to disguise their traffic for sufficiently long connections.

Each of these techniques creates a time series of a certain flow characteristic and uses it to compare flow pairs. This implies a pairwise comparison over each value of the time series. It also means that the stepping stone detection algorithms rely heavily on the accuracy of series of one flow characteristic value.

Because of the one-to-many "multicasting" model of the C2 (and chat) architecture, we expect the communication flows between the botnet C2 host and the IRC server, and between the IRC server and the botnet members, to be temporally correlated. Since data sent to the chat server is promptly multicast to all chat members, the flows to (and from) all chat members should exhibit similar timing characteristics as well as contemporary fluctuations in bandwidth.

Any of the flow correlation algorithms based on temporal flow characteristics cited above could be applied to this stage, but they are each computationally expensive. These and most other current flow correlation algorithms examine each flow every time there is a new packet arrival, and every pairwise "correlation value" is updated. This implies O(n^2) calculations for each packet, where n is the number of active flows. We prefer an algorithm that performs a calculation only once per packet arrival — to update that packet's flow value — delaying the O(n^2) comparison until the time when flow correlation question was asked. We developed such an algorithm for use in stepping stone detection [27]. This algorithm uses multiple flow characteristics but remains efficient in per-flow correlation value updating.

5.2 Multi-Dimensional Flow Correlation

In constructing a new flow correlation algorithm, our first aim is to increase robustness by including more than one flow characteristic for comparison. Our second aim is to record the time series of the values of these characteristics more efficiently and

eliminate the need for maintaining a full correlation matrix over all time. Let us look at the second aim first.

Time series are arbitrarily long time-value pairs that are not easy to manipulate. Statistical measures over the time series, however, attempt to describe the shape of the data in a finite space, and are much easier to manage. Taking the average, for example, describes an arbitrarily long series of values in one value, but at the loss of a lot of fidelity. Taking the second moment, the variance, gets some of that fidelity back by describing how different the values are from each other. Further moments describe the peakedness of the data (kurtosis) and the symmetry of the peaks (skew).

A nice aspect of using moments is that they can be estimated on the fly, and any new event causes the recalculation of the moments for that flow only. So a characteristic of a flow — say packet sizes — can be described by a small vector of statistical moments of that characteristic. This satisfies part of the second aim for efficient recording of the values for the flow characteristics.

If a single characteristic for a flow can be described using a small vector, then why not widen the vector to include statistical moments for other flow characteristics? Doing this would satisfy the first aim of including multiple characteristics in a flow correlation algorithm, but it does not suggest how to combine the multiple characteristics into a single comparison.

Our answer is to treat each flow's description vector as a point in n-space, where n is the cardinality of the vector, and apply a distance calculation as a measure of correlation, where nearness is more correlated. The distance does not have to be maintained for all flow pairs over all time, but calculated only when the correlation question is raised. This satisfies the second part of aim two.

Expressing a time series as a set of moments loses fidelity, which means that some unrelated flows with different time series of values over a particular characteristic might accidentally have the same moments over that time series. This is a matter of entropy; if there /indexentropy is not enough descriptive power in the vector, the flows cannot be adequately distinguished one flow from another, and false positives will occur. Our hypothesis is that, by adding more characteristics, the entropy is raised, mitigating the loss of fidelity of reducing any one characteristic to a vector of moments.

Determining the Characteristics

We have been abstractly discussing the use of multiple flow characteristics in a flow correlation algorithm, but determining which characteristics are most useful is the subject of studies and experiments. However, there are some useful features in a flow characteristic that might make one better suited than another.

First, the characteristic should be dynamic and expressed as a time series. Samples of the moments of a dynamic data set are themselves dynamic. Two flows that share this dynamic nature of the moments are likely to be correlated. If the moments remain static, then two uncorrelated flows with the same values will always show as a false positive.

Next, the characteristic should measure something about the flow that is imposed externally, not by the communications protocol. Since TCP/IP is probably the common transport, then characteristics imposed by TCP or IP will likely not discriminate between flows. Packet size is an example of a bad characteristic when the application gives TCP/IP a very large amount of data to send, but it is a good one when the application offers small amounts of data. Packet inter-arrival times and packet inter-burst times are similar.

Finally, for practical purposes, the characteristic should be easily measured. Throughput, for example, requires maintaining an amount of data seen over a window of time, while packet arrival times require no history.

Estimating the Moments

Since the time series values are arbitrarily long, and the are arriving in real time, we need to calculate the moments as a running estimate. The estimated weighted moving average (EWMA) is a nice way to estimate an average while weighting the influence of the past. The formula is: $\mathrm{newEWMA} = \alpha(\mathrm{newValue}) + (1-\alpha)(\mathrm{oldEWMA})$. We set α at 0.75 to emphasis new events while maintaining the smoothing effect of old events. The second moment, variance, is estimated in a similar fashion: $\mathrm{newVAR} = \alpha(|\mathrm{newValue} - \mathrm{EWMA}|) + (1-\alpha)(\mathrm{oldVAR})$. We do not use higher moments.

Calculating the Distance

We treat a flow's vector of characteristics as a point in n-space, and use a distance measure to determine correlation based on closeness. But values from different characteristics, and from different moments within each characteristic, have magnitudes that must be normalized before they can be used, otherwise characteristics with large values will artificially outweigh characteristics with smaller values. Further, some characteristics can have unbounded values.

Rather than normalize values and then use them to find the distance, it is better to normalize the difference. This way we maintain the natural meaning of the difference of v_1 and v_2, then fit that into a 0-to-1 scale.

One common normalizer is an exponential: $\mathrm{norm\_diff} = 1^{e-\lambda}(|v_1 - v_2|)$, where λ is a weighting factor to determine how steeply the asymptote rises to 1. It makes sense that each characteristic would need a different λ, but if λ is set incorrectly, there will be too much or too little distinction between values of $v_1 - v_2$.

Instead, we use the following: $\mathrm{norm\_diff} = (|v_1 - v_2|)/(v_1 + v_2)$. As v_2 approaches v_1 from below, the normalized difference drops off nearly linearly. As v_2 grows larger than v_1, the normalized difference grows asymptotically to 1. This normalizer is self-weighting and does not require special values such as λ.

The distance between two flows is calculated using the Euclidean formula of taking the square root of the sum of the squares of the differences:

$$\mathrm{distance} = \sqrt{\sum_{i=1}^{n} (\mathrm{norm\_diff}_i)^2}$$

where n is the number of values in the flow characteristics vector, and $\mathrm{norm\_diff}_i$ is the normalized difference of the i^{th} value in the vector. Since each vector element difference is normalized to 1, the maximum distance is $\sqrt{n}$.

5.3 Correlation Results

Figures 6 and 7 display the results of pairwise distances between each of the 95 filtered flows. (Because the classification stage dropped some of the ground-truth botnet flows, we ran the correlation algorithm over the filtered, but not classified, flows.) Figure 6 clearly shows a horizontal band of flow pairs whose Euclidean distance is very small, separated by a band of white space up to distance of about 2. This indicates that a group of flows are clustered very near each other in n-space, and that there is a gap between that cluster and the next nearest flows.

Figure 7 also shows this gap in terms of a probability distribution of the distances. Note that there is a substantial spike near distance 0, then there is a flat area (no or few flow pairs) until distance 2. The spike is a cluster of flow pairs that are very close in distance. In fact, there are 9 flow pairs whose distance is less than 0.5, and it is this set that forms the cluster of interest.

The identification of clusters of correlated flows certainly suggest further investigation, which is the aim of the next stage, the topological analysis. This correlation stage does not prove the existence of a botnet — there is no test for maliciousness in the filtering, classifying, and clustering of flows — but given a cluster of flows, the natural next question is, What structure do these and other flows form, and does this structure identify a host that is acting like a botnet controller.

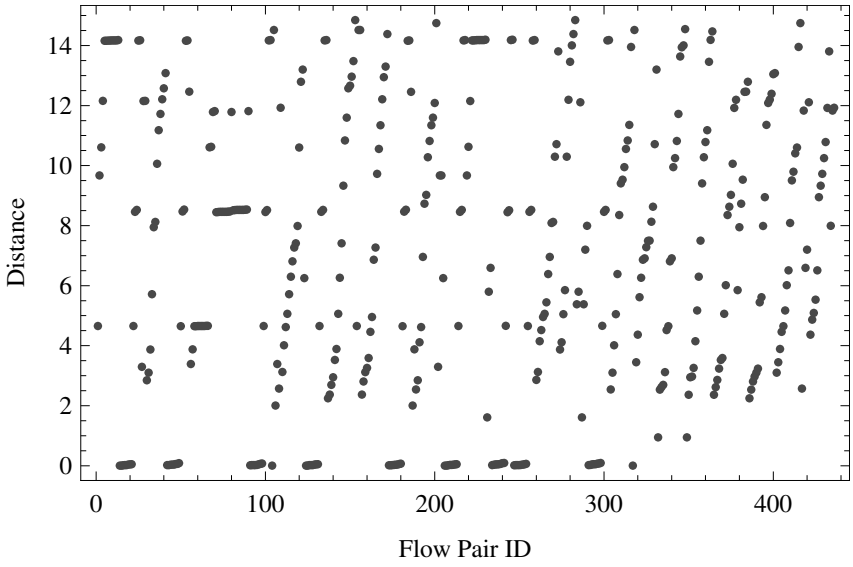

Fig. 6. Scatter Plot of Distances between Flow Pairs

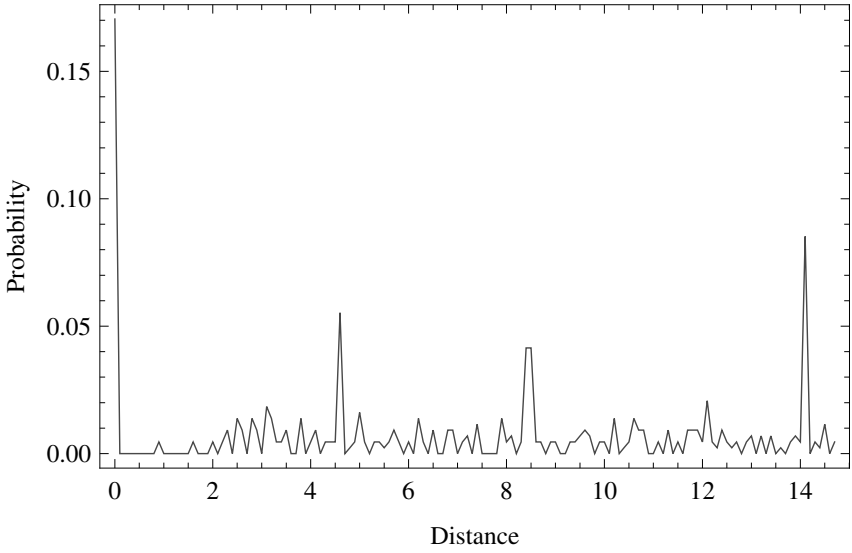

Fig. 7. Distance Probability Density Function of Flow Pair Distances

6 Topological Analysis Stage

The topological analysis starts by selecting only those flow pairs that are highly correlated. Figures 6 and 7 both show that there is a grouping of highly correlated flow pairs with distances close to 0. Our hypothesis is that these highly correlated flow pairs correspond to botnet C2 flows. We isolate these flow pairs by selecting only those flow pairs with a distance of less than 0.5. These flow pairs correspond to the top 17% most highly correlated flow pairs. On further investigation, we note that every one of these flow pairs corresponds to a C2 connection between a zombie host and the rendezvous point (IRC server), thus validating our hypothesis.

The next step in the topological analysis is to analyze the overall correlation structure of the correlated flow pairs. This process can be easily automated. Figure 8 shows a graph where each node corresponds to a unique flow pair identifier and each edge connects two highly correlated flow pairs. The graph shows a "perfect" or mesh clustering between the set of nine highly correlated flow pairs. This perfect clustering shows that each of the highly correlated flow pairs correlates with all of the other highly correlated flow pairs. In other words, the nine botnet C2 connections all correlate extremely well with each other. This again confirms our hypothesis.

The final step in the topological analysis is to determine the communication topology that corresponds to these highly correlated flow pairs and to identify which of the hosts, if any, is acting as a rendezvous point. This is a two part process that can be automated easily. First, we generate a graph that has as its edges the highly correlated flow pairs identified in the first step of the topological analysis and as its nodes the host IP addresses that correspond to the endpoints of these flow pairs. Second, we look for the node with the highest in-degree or out-degree and select that as

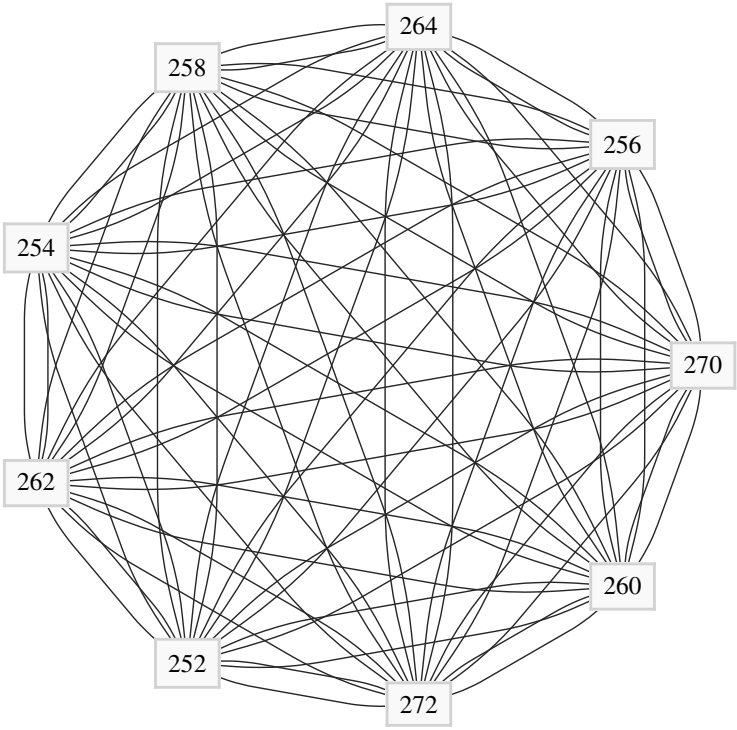

Fig. 8. Flow Pair Clustering

a candidate rendezvous point (IRC Server). Figure 9 shows a directed graph generated using the first part of this procedure (in this figure, IP addresses have been replaced by labels to identify the roles of the hosts). The communication structure of the botnet is immediately obvious from the figure and it is very easy to identify the rendezvous point as the node having the highest in-degree.

The topological analysis is able to identify nine out of the ten zombie hosts in our botnet. The nine zombies identified correspond to "local" zombies that are all located on machines in the same building at BBN (see Figure 3). The one zombie host not identified corresponds to a "remote" bot running on an offsite host. This result is perfectly understandable: we would not expect flows from a remote bot to correlate that well with flows from local bots as the difference in communication paths would almost always result in significant differences in flow characteristics.

In summary, the topological analysis stage examines the structure of highly correlated flow pairs. By constructing graphs of these correlated flow pairs, graphs of the corresponding node pairs and then looking for nodes with high in-degree, it is

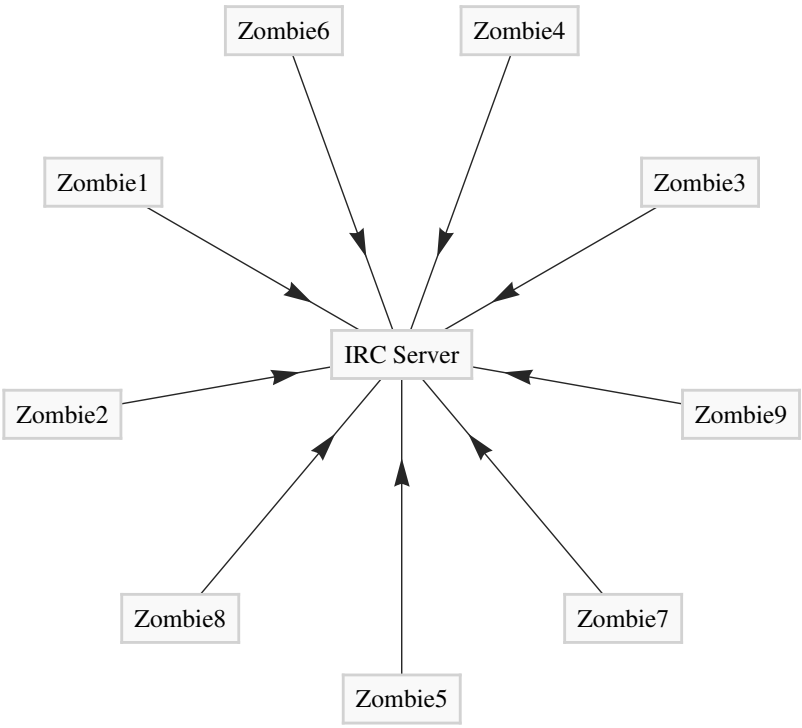

Fig. 9. Host-based Clustering

possible to identify the communication structure of our botnet, the rendezvous point and nine out of ten zombies. The results from topological analysis stage clearly supported our hypothesis that C2 botnet flows are highly correlated.

7 Discussion

While it has been suggested that botnet controllers will migrate from IRC as their preferred C2 infrastructure [25], the abstract model of tight central control represented by IRC is very efficient and will likely survive for quite some time. It is important, therefore, to consider a system that detects very large, high volume data sets for evidence of tight botnet C2 activity.

Our system performs gross, simple filtering to reduce the amount of data that will be subjected to more computationally intensive algorithms. Once the data has been filtered, the flows are classified using machine learning techniques, then the flows that are in the “chat” class are correlated to find clusters of flows that share similar

timing and packet size characteristics. The cluster is then analyzed to try to identify the botnet controller host.

Our experiment with Dartmouth campus data, starting with nearly 9 million flows augmented with traffic traces from a benign botnet, shows that the ground truth botnet C2 flows can indeed survive the data reduction and correlation to be identified as a cluster. These results show that the method is promising.

This method is also nicely suited for real-time analysis of traffic data. The filtering stage requires very simple logic to cull the data set down by a factor of 37. While we may not be able to expect that degree of reduction in all cases, there was nothing particularly special about the Dartmouth data that contributed to the reduction factor. The culling of the data, especially when done in real time, allows much more time for more complex algorithms later in the pipe, namely the machine learning classifiers and the correlation.

An important lesson learned from our classification stage is the importance of both legitimate and malicious training traffic and an accurate manner to label it. Given such representative training traffic, machine learning-based classifiers can perform well and be very effective. The trick is to get a good training set.

Our experience with the new correlation algorithm showed that the algorithm holds promise. The algorithm we used is designed to reduce the computational complexity of comparing n flows in a pairwise manner. The resulting cluster, while not a complete set of flows from the ground truth botnet, was certainly enough to allow the topological analysis of the flow endpoints, and the rest of the ground-truth botnet traffic was easily extracted.

Detecting botnet activity is presently labor intensive and largely *ad hoc*. Our pipelined botnet C2 detection system shows that it is possible to comb through packet traces, even in real time, to extract evidence of tight command and control activity and, from that evidence, discover the botnet controller.

Acknowledgments

This work was sponsored by the U.S. Army Research Office under contract No. W911NF-05-C-0066. The content of the information does not necessarily reflect the position or the policy of the U.S. Government, and no official endorcement should be inferred.

The authors wish to thank Doug Maughan and Cliff Wang for their support, and Mark Allman for his valuable insights. We also thank David Kotz and gratefully acknowledge the use of wireless data from the CRAWDAD archive at Dartmouth College. We also wish to acknowledge the support and contributions of our colleagues at BBN Technologies: Christine Jones, Beverly Schwartz, Sarah Edwards, Walter Milliken, and Alden Jackson.

References

1. US-CERT Vulnerability Notes Database. http://www.kb.cert.org/vuls/.

2. Paul Barford and Vinod Yegneswaran. An inside look at botnets (to appear in series: Advances in information security, springer), 2006.
3. A. Blum, D. Song, and S. Venkataraman. Detection of interactive stepping stones: Algorithms and confidence bounds. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID '04)*, September 2004.
4. David Dagon, Cliff Zou, and Wenke Lee. Modeling botnet propagation using time zones. In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS '06)*, February 2006.
5. Defense Security Service. Memorandum for facility security officers: Foreign-based threat to defense contractor unclassified networks, October 18, 2005.
6. Christian Dewes, Arne Wichmann, and Anja Feldmann. An analysis of internet chat systems. In *IMC '03: Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 51–64, New York, NY, USA, 2003. ACM Press.
7. David L. Donoho, Ana Georgina Flesia, Umesh Shankar, Vern Paxson, Jason Coit, and Stuart Staniford. Multiscale stepping-stone detection: Detecting pairs of jittered interactive streams by exploiting maximum tolerable delay. In *Proc. International Symposium on Recent Advances in Intrusion Detection*, pages 17–35, October 2002.
8. Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley & Sons, Inc., 2 edition, 2001.
9. T. He and L. Tong. Detecting encrypted stepping-stone connections. *IEEE Transactions on Signal Processing*, 2007.
10. Thorsten Holz. A Short Visit to the Bot Zoo. *IEEE Security & Privacy*, 3(3):76–79, May 2005.
11. Kevin J. Houle and George M. Weaver. Trends in denial of service technology. CERT Coordination Center, October 2001.
12. A. Householder, Art Manion, Linda Pesante, George M. Weaver, and Rob Thomas. Managing the threat of denial-of-service attacks. CERT Coordination Center, October 2001.
13. S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-sale: Surviving organized ddos attacks that mimic flash crowds. In *Proceedings of the 2nd Symposium on Networked Systems Design and Implementation*, May 2005.
14. Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-scale botnet detection and characterization. In *Proceedings of the First Workshop on Hot Topics in Understanding Botnets*, April 2007.
15. David Kotz and Tristan Henderson. CRAWDAD: A Community Resource for Archiving Wireless Data at Dartmouth. *IEEE Pervasive Computing*, 4(4), oct-dec 2006.
16. Elias Levy. The Making of a Spam Zombie Army. *IEEE Security & Privacy*, 1(4):58–59, July 2003.
17. Carl Livadas, Robert Walsh, David Lapsley, and W. Timothy Strayer. Using Machine Learning Techniques to Identify Botnet Traffic. In *Proceedings of the 2nd IEEE LCN Workshop on Network Security*, 2006.
18. Bill McCarty. Automated Identity Theft. *IEEE Security & Privacy*, 1(5):89–92, September 2003.
19. Bill McCarty. Botnets: Big and Bigger. *IEEE Security & Privacy*, 1(4):87–90, July 2003.
20. Andrew W. Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 50–60, New York, NY, USA, 2005. ACM Press.
21. R. Naraine. Botnet hunters search for 'command and control' servers. *eWeek*, June 17, 2005.

22. National Infrastructure Security Coordination Center. Targeted trojan email attacks. NISCC Briefing 08/2005, June 16, 2005.
23. Anirudh Ramachandran, Nick Feamster, and David Dagon. Revealing botnet membership using DNSBL counter-intelligence. In *Proceedings of the 2nd Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, 2006.
24. Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 135–148, New York, NY, USA, 2004. ACM Press.
25. Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, pages 512–521, New York, NY, USA, 2004. ACM Press.
26. Alex C. Snoeren, Craig Partridge, Luis A. Sanchez, Christine E. Jones, Fabrice Tchakountio, Beverly Schwartz, Stephen T. Kent, and W. Timothy Strayer. Single-packet IP traceback. *ACM/IEEE Trans. on Networking*, December 2002.
27. W. Timothy Strayer, Christine Jones, Beverley Schwartz, Sarah Edwards, Walter Milliken, and Alden Jackson. Efficient multi-dimensional flow correlation. In *Proceedings of the 32st IEEE Conference on Local Computer Networks (LCN'07)*, November 2007. Submitted for publication.
28. W. Timothy Strayer, Christine Jones, Beverly Schwartz, Joanne Mikkelson, and Carl Livadas. Architecture for Multi-Stage Network Attack Traceback. In *Proceedings of the IEEE LCN Workshop on Network Security (WoNS 2005)*, Sydney, Australia, November 2005.
29. W. Timothy Strayer, Robert Walsh, Carl Livadas, and David Lapsley. Detecting Botnets with Tight Command and Control. In *Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN'06)*, November 2006.
30. Symantec. Symantec Internet Security Threat Report. Trends for July – December 06, March 2007.
31. The Honeynet Project. *Know Your Enemy : Learning about Security Threats*. Addison-Wesley Professional; 2 edition (May 17, 2004), March 2004.
32. Rob Thormeyer. Hacker arrested for breaching dod systems with 'botnets'. *Government Computer News*, November 4, 2005.
33. Xinyuan Wang, Douglas S. Reeves, and S. Felix Wu. Inter-packet delay based correlation for tracing encrypted connections through stepping stones. In *Proc. European Symposium on Research in Computer Security*, pages 244–263, October 2002.
34. Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques (2nd Edition)*. Morgan Kaufmann, San Francisco, CA, 2005.
35. Kunikazu Yoda and Hiroaki Etoh. Finding a connection chain for tracing intruders. In *Proc. European Symposium on Research in Computer Security*, pages 191–205, October 2000.
36. L. Zhang, A. G. Persaud, A. Johnson, and Y. Guan. Detection of stepping stone attacks under delay and chaff perturbations. In *Proceedings of the 25th IEEE International Performance Computing and Communications Conference*, April 2006.
37. Yin Zhang and Vern Paxson. Detecting stepping stones. In *Proc. USENIX Security Symposium '00*, pages 171–184, August 2000.