# BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection


**Guofei Gu[1], Roberto Perdisci[2], Junjie Zhang[1], and Wenke Lee[1]**
**[1]College of Computing, Georgia Institute of Technology**
**[2]Damballa, Inc. Atlanta, GA 30308, USA**
{guofei,jjzhang,wenke}@cc.gatech.edu, perdisci@damballa.com

**Abstract:**

Botnets are now the key platform for many Internet attacks, such as spam, distributed denial-of-service (DDoS), identity theft, and phishing. Most of the current botnet detection approaches work only on specific botnet command and control (C&C) protocols (e.g., IRC) and structures (e.g., centralized), and can become ineffective as botnets change their C&C techniques. In this paper, we present a general detection framework that is independent of botnet C&C protocol and structure, and requires no *a priori* knowledge of botnets (such as captured bot binaries and hence the botnet signatures, and C&C server names/addresses). We start from the definition and essential properties of botnets. We define a botnet as a *coordinated group* of *malware* instances that are *controlled* via C&C communication channels. The essential properties of a botnet are that the bots communicate with some C&C servers/peers, perform malicious activities, and do so in a similar or correlated way. Accordingly, our detection framework clusters similar communication traffic and similar malicious traffic, and performs cross cluster correlation to identify the hosts that share both similar communication patterns *and* similar malicious activity patterns. These hosts are thus bots in the monitored network. We have implemented our BotMiner prototype system and evaluated it using many real network traces. The results show that it can detect real-world botnets (IRC-based, HTTP-based, and P2P botnets including Nugache and Storm worm), and has a very low false positive rate.

# 1 Introduction

Botnets are becoming one of the most serious threats to Internet security. A botnet is a network of compromised machines under the influence of malware (bot) code. The botnet is commandeered by a ``botmaster'' and utilized as ``resource'' or ``platform'' for attacks such as distributed denial-of-service (DDoS) attacks, and fraudulent activities such as spam, phishing, identity theft, and information exfiltration.

In order for a botmaster to command a botnet, there needs to be a command and control (C&C) channel through which bots receive commands and coordinate attacks and fraudulent activities. The C&C channel is the means by which individual bots form a bot*net*. Centralized C&C structures using the Internet Relay Chat (IRC) protocol have been utilized by botmasters for a long time. In this architecture, each bot logs into an IRC channel, and seeks commands from the botmaster. Even today, many botnets are still designed this way. Quite a few botnets, though, have begun to use other protocols such as HTTP [24,8,14,39], probably because HTTP-based C&C communications are more stealthy given that Web traffic is generally allowed in most networks. Although centralized C&C structures are effective, they suffer from the single-point-of-failure problem. For example, if the IRC channel (or the Web server) is taken down due to detection and response efforts, the botnet loses its C&C structure and becomes a collection of isolated compromised machines. Recently, botmasters began using peer-to-peer (P2P) communication to avoid this weakness. For example, Nugache [28] and Storm worm [18,23] (a.k.a. Peacomm) are two representative P2P botnets. Storm, in particular, distinguishes itself as having infected a large number of computers on the Internet and effectively becoming one of the ``world's top super-computers'' [27] for the botmasters.

Researchers have proposed a few approaches [7,17,19,20,26,29,35,40] to detect the existence of botnets in monitored networks. Almost all of these approaches are designed for detecting botnets that use IRC or HTTP based C&C [7,17,26,29,40]. For example, Rishi [17] is designed to detect IRC botnets using known IRC bot nickname patterns as signatures. In [26,40], network flows are clustered and classified according to IRC-like traffic patterns. Another more recent system, BotSniffer, [20] is designed mainly for detecting C&C activities with centralized servers (with protocols such as IRC and HTTP[1]). One exception is perhaps BotHunter [19], which is capable of detecting bots regardless of the C&C structure and network protocol as long as the bot behavior follows a *pre-defined* infection life cycle dialog model.

However, botnets are evolving and can be quite flexible. We have witnessed that the protocols used for C&C evolved

from IRC to others (e.g., HTTP [24,8,14,39]), and the structure moved from centralized to distributed (e.g., using P2P [28,18]). Furthermore, a botnet during its lifetime can also change its C&C server address frequently, e.g., using fast-flux service networks [22]. Thus, the aforementioned detection approaches designed for IRC or HTTP based botnets may become ineffective against the recent/new botnets. Even BotHunter may fail as soon as botnets change their infection model(s).

Therefore, we need to develop a next generation botnet detection system, which should be independent of the C&C protocol, structure, and infection model of botnets, and be resilient to the change of C&C server addresses. In addition, it should require no *a priori* knowledge of specific botnets (such as captured bot binaries and hence the botnet signatures, and C&C server names/addresses).

In order to design such a general detection system that can resist evolution and changes in botnet C&C techniques, we need to study the *intrinsic* botnet communication and activity characteristics that remain detectable with the proper detection features and algorithms. We thus start with the definition and essential properties of a botnet. We define a botnet as:

``A *coordinated group* of *malware* instances that are *controlled* via C&C channels''.

The term ``malware'' means these bots are used to perform *malicious activities*. According to [44], about 53% of botnet activity commands observed in thousands of real-world IRC-based botnets are related to scan (for the purpose of spreading or DDoS[2]), and about 14.4% are related to binary downloading (for the purpose of malware updating). In addition, most of HTTP-based and P2P-based botnets are used to send spam [18,39]. The term ``controlled'' means these bots have to contact their C&C servers to obtain commands to carry out activities, e.g., to scan. In other words, there should be *communication between bots and C&C servers/peers* (which can be centralized or distributed). Finally, the term ``coordinated group'' means that multiple (at least two) bots within the same botnet will perform *similar* or *correlated* C&C communications and malicious activities. If the botmaster commands each bot individually with a different command/channel, the bots are nothing but some isolated/unrelated infections. That is, they do not function as a bot*net* according to our definition and are out of the scope of this work[3].

We propose a general detection framework that is based on these essential properties of botnets. This framework monitors both *who is talking to whom* that may suggest C&C communication activities and *who is doing what* that may suggest malicious activities, and finds a *coordinated group pattern* in both kinds of activities. More specifically, our detection framework clusters similar communication activities in the *C-plane* (C&C communication traffic), clusters similar malicious activities in the *A-plane* (activity traffic), and performs cross cluster correlation to identify the hosts that share both similar communication patterns *and* similar malicious activity patterns. These hosts, according to the botnet definition and properties discussed above, are bots in the monitored network.

This paper makes the following main contributions.

- We develop a novel general botnet detection framework that is grounded on the definition and essential properties of botnets. Our detection framework is thus independent of botnet C&C protocol and structure, and requires no *a priori* knowledge (e.g., C&C addresses/signatures) of specific botnets. It can detect both centralized (e.g., IRC,HTTP) and current (and possibly future) P2P based botnets.
- We define a new ``aggregated communication flow'' (C-flow) record data structure to store aggregated traffic statistics, and design a new layered clustering scheme with a set of traffic features measured on the C-flow records. Our clustering scheme can accurately and efficiently group similar C&C traffic patterns.
- We build a BotMiner prototype system based on our general detection framework, and evaluate it with multiple real-world network traces including normal traffic and several real-world botnet traces that contain IRC, HTTP and P2P-based botnet traffic (including Nugache and Storm). The results show that BotMiner has a high detection rate and a low false positive rate.

The rest of the paper is organized as follows. In Section 2, we describe the assumptions, objectives, architecture of our BotMiner detection framework, and its detection algorithms and implementation. In Section 3, we describe our evaluation on various real-world network traces. In Section 4, we discuss current limitations and possible solutions. We review the related work in Section 5 and conclude in Section 6.

# 2 BotMiner Detection Framework and Implementation

## 2.1 Problem Statement and Assumptions

According to the definition given above, a botnet is characterized by both a C&C communication channel (from which the botmaster's commands are received) and malicious activities (when commands are executed). Some other forms of malware (e.g., worms) may perform malicious activities, but they do not connect to a C&C channel. On the other hand, some normal applications (e.g., IRC clients and normal P2P file sharing software) may show communication patterns similar to a botnet's C&C channel, but they do not perform malicious activities.

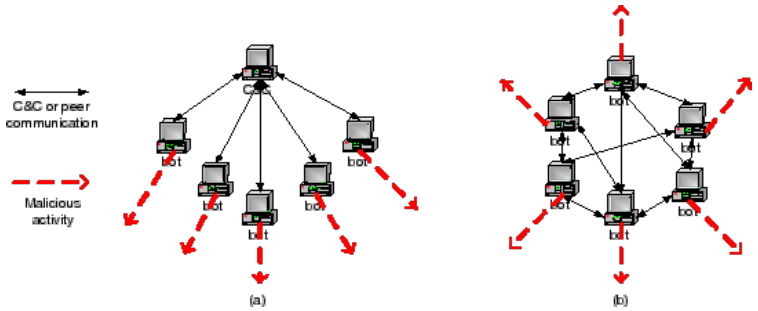


**Figure 1:** Possible structures of a botnet: (a) centralized; (b) peer-to-peer.

Figure 1 illustrates two typical botnet structures, namely *centralized* and *P2P*. The bots receive commands from the botmaster using a *push* or *pull* mechanism [20] and execute the assigned tasks.

The operation of a centralized botnet is relatively easy and intuitive [20], whereas this is not necessarily true for P2P botnets. Therefore, here we briefly illustrate an example of a typical P2P-based botnet, namely Storm worm [18,23]. In order to issue commands to the bots, the botmaster publishes/shares command files over the P2P network, along with specific search keys that can be used by the bots to find the published command files. Storm bots utilize a pull mechanism to receive the commands. Specifically, each bot frequently contacts its neighbor peers searching for specific keys in order to locate the related command files. In addition to search operations, the bots also frequently communicate with their peers and send *keep-alive* messages.

In both centralized and P2P structures, bots within the same botnet are likely to behave similarly in terms of communication patterns. This is largely due to the fact that bots are non-human driven, pre-programmed to perform the same routine C&C logic/communication as coordinated by the same botmaster. In the centralized structure, even if the address of the C&C server may change frequently (e.g., by frequently changing the A record of a Dynamic DNS domain name), the C&C communication patterns remain unchanged. In the case of P2P-based botnets, the peer communications (e.g., to search for commands or to send *keep-alive* messages) follow a similar pattern for all the bots in the botnet, although each bot may have a different set of neighbor peers and may communicate on different ports.

Regardless of the specific structure of the botnet (centralized or P2P), members of the same botnet (i.e., the bots) are coordinated through the C&C channel. In general, a bot*net* is different from a set of *isolated* individual malware instances, in which each different instance is used for a totally different purpose. Although in an extreme case a botnet can be configured to degenerate into a group of *isolated hosts*, this is not the common case. In this paper, we focus on the most typical and useful situation in which bots in the same bot*net* perform similar/coordinated activities. To the best of our knowledge, this holds true for most of the existing botnets observed in the wild.

To summarize, we assume that bots within the same botnet will be characterized by similar malicious activities, as well as similar C&C communication patterns. Our assumption holds even in the case when the botmaster chooses to divide a botnet into *sub-botnets*, for example by assigning different tasks to different sets of bots. In this case, each sub-botnet will be characterized by similar malicious activities and C&C communications patterns, and our goal is to detect each sub-botnet. In Section 4 we provide a detailed discussion on possible *evasive* botnets that may violate our assumptions.

## 2.2 Objectives

The objective of BotMiner is to detect *groups* of compromised machines within a monitored network that are part of a botnet. We do so by passively analyzing network traffic in the monitored network.

Note that we do not aim to detect botnets at the very moment when victim machines are compromised and infected

with malware (bot) code. In many cases these events may not be observable by passively monitoring network traffic. For example, an already infected laptop may be carried in and connected to the monitored network, or a user may click on a malicious email attachment and get infected. In this paper we are not concerned with the way internal hosts become infected (e.g., by malicious email attachments, remote exploiting, and Web drive-by download). We focus on the detection of groups of already compromised machines inside the monitored network that are part of a botnet.

Our detection approach meets several goals:

- it is independent of the protocol and structure used for communicating with the botmaster (the C&C channel) or peers, and is resistant to changes in the location of the C&C server(s).
- it is independent of the content of the C&C communication. That is, we do not inspect the content of the C&C communication itself, because C&C could be encrypted or use a customized (obscure) protocol.
- it generates a low number of false positives and false negatives.
- the analysis of network traffic employs a reasonable amount of resources and time, making detection relatively efficient.

## 2.3 Architecture of BotMiner Detection Framework

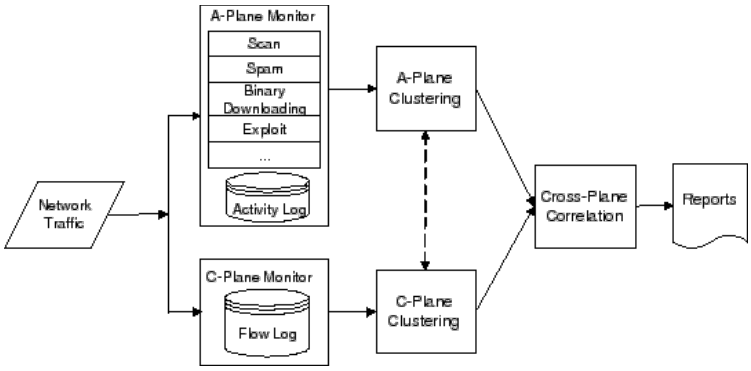


**Figure 2:** Architecture overview of our BotMiner detection framework.

Figure 2 shows the architecture of our BotMiner detection system, which consists of five main components: C-plane monitor, A-plane monitor, C-plane clustering module, A-plane clustering module, and cross-plane correlator.

The two traffic monitors in C-plane and A-plane can be deployed at the edge of the network examining traffic between internal and external networks, similar to BotHunter [19] and BotSniffer [20][4]. They run in parallel and monitor the network traffic. The C-plane monitor is responsible for logging network flows in a format suitable for efficient storage and further analysis, and the A-plane monitor is responsible for detecting suspicious activities (e.g., scanning, spamming, and exploit attempts). The C-plane clustering and A-plane clustering components process the logs generated by the C-plane and A-plane monitors, respectively. Both modules extract a number of features from the raw logs and apply clustering algorithms in order to find groups of machines that show very similar communication (in the C-plane) and activity (in the A-plane) patterns. Finally, the cross-plane correlator combines the results of the C-plane and A-plane clustering and makes a final decision on which machines are possibly members of a botnet. In an ideal situation, the traffic monitors should be distributed on the Internet, and the monitor logs are reported to a central repository for clustering and cross-plane analysis.

In our current prototype system, traffic monitors are implemented in C for the purpose of efficiency (working on real-time network traffic). The clustering and correlation analysis components are implemented mainly in Java and R (http://www.r-project.org/), and they work offline on logs generated from the monitors.

The following sections present the details of the design and implementation of each component of the detection framework.

## 2.4 Traffic Monitors

**C-plane Monitor.**

The C-plane monitor captures network flows and records information on *who is talking to whom*. Many network routers support the logging of network flows, e.g., Cisco (www.cisco.com) and Juniper (www.juniper.net) routers. Open source solutions like Argus (Audit Record Generation and Utilization System, http://www.qosient.com/argus) are also available. We adapted an efficient network flow capture tool developed at our research lab, i.e., `fcapture` [5], which is based on the Judy library (http://judy.sourceforge.net/). Currently, we limit our interest to TCP and UDP flows. Each flow record contains the following information: time, duration, source IP, source port, destination IP, destination port, and the number of packets and bytes transfered in both directions. The main advantage of our tool is that it works very efficiently on high speed networks (very low packet loss ratio on a network with 300Mbps traffic), and can generate very compact flow records that comply with the requirement for further processing by the C-plain clustering module. As a comparison, our flow capturing tool generates compressed records ranging from 200MB to 1GB per day from the traffic in our academic network, whereas Argus generates around 36GB of compressed binary flow records per day on average (without recording any payload information). Our tool makes the storage of several weeks or even months of flow data feasible.

**A-plane Monitor.**

The A-plane monitor logs information on *who is doing what*. It analyzes the outbound traffic through the monitored network and is capable of detecting several malicious activities that the internal hosts may perform. For example, the A-plane monitor is able to detect scanning activities (which may be used for malware propagation or DoS attacks), spamming, binary downloading (possibly used for malware update), and exploit attempts (used for malware propagation or targeted attacks). These are the most common and ``useful'' activities a botmaster may command his bots to perform [9,33,44].

Our A-plane monitor is built based on Snort [36], an open-source intrusion detection tool, for the purpose of convenience. We adapted existing intrusion detection techniques and implemented them as Snort pre-processor plug-ins or signatures. For scan detection we adapted SCADE (Statistical sCan Anomaly Detection Engine), which is a part of BotHunter [19] and available at [11]. Specifically, we mainly use two anomaly detection modules: the *abnormally-high scan rate* and weighted *failed connection rate*. We use an `OR` combination rule, so that an event detected by either of the two modules will trigger an alert. In order to detect spam-related activities, we developed a new Snort plug-in. We focused on detecting anomalous amounts of DNS queries for MX records from the same source IP and the amount of SMTP connections initiated by the same source to mail servers outside the monitored network. Normal clients are unlikely to act as SMTP servers and therefore should rely on the internal SMTP server for sending emails. Use of many distinct external SMTP servers for many times by the same internal host is an indication of possible malicious activities. For the detection of PE (Portable Executable) binary downloading we used an approach similar to PEHunter [42] and BotHunter's egg download detection method [19]. One can also use specific exploit rules in BotHunter to detect internal hosts that attempt to exploit external machines. Other state-of-the-art detection techniques can be easily added to our A-plane monitoring to expand its ability to detect typical botnet-related malicious activities.

It is important to note that A-plane monitoring alone is not sufficient for botnet detection purpose. First of all, these A-plane activities are not exclusively used in botnets. Second, because of our relatively loose design of A-plane monitor (for example, we will generate a log whenever there is a PE binary downloading in the network regardless of whether the binary is malicious or not), relying on only the logs from these activities will generate a lot of false positives. This is why we need to further perform A-plane clustering analysis as discussed shortly in Section 2.6.

## 2.5 C-plane Clustering

C-plane clustering is responsible for reading the logs generated by the C-plane monitor and finding clusters of machines that share similar communication patterns. Figure 3 shows the architecture of the C-plane clustering.

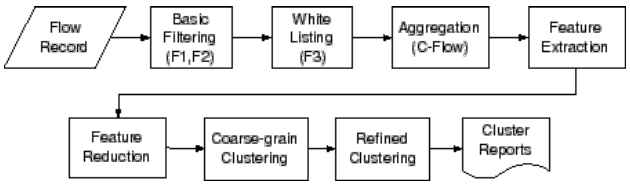


**Figure 3:** C-plane clustering.

First of all, we filter out irrelevant (or uninteresting) traffic flows. This is done in two steps: basic-filtering and white-listing. It is worth noting that these two steps are not critical for the proper functioning of the C-plane clustering module. Nonetheless, they are useful for reducing the traffic workload and making the actual clustering process more efficient. In the basic-filtering step, we filter out all the flows that are not directed from internal hosts to

external hosts. Therefore, we ignore the flows related to communications between internal hosts[6] and flows initiated from external hosts towards internal hosts (filter rule 1, denoted as F1). We also filter out flows that are not completely established (filter rule 2, denoted as F2), i.e., those flows that only contain one-way traffic. These flows are mainly caused by scanning activity (e.g., when a host sends SYN packets without completing the TCP hand-shake). In white-list filtering, we filter out those flows whose destinations are well known as legitimate servers (e.g., Google, Yahoo!) that will unlikely host botnet C&C servers. This filter rule is denoted as F3. In our current evaluation, the white list is based on the US top 100 and global top 100 most popular websites from Alexa.com.

After basic-filtering and white-listing, we further reduce the traffic workload by aggregating related flows into communication flows (C-flows) as follows. Given an epoch $E$ (typically one day), all $m$ TCP/UDP flows that share the same protocol (TCP or UDP), source IP, destination IP and port, are aggregated into the same C-flow $c_i = \{f_j\}_{j=1..m}$, where each $f_j$ is a single TCP/UDP flow. Basically, the set $\{c_i\}_{i=1..n}$ of all the $n$ C-flows observed during $E$ tells us ``who was talking to whom'', during that epoch.

### 2.5.1 Vector Representation of C-flows

The objective of C-plane clustering is to group hosts that share similar communication flows. This can be accomplished by clustering the C-flows. In order to apply clustering algorithms to C-flows we first need to translate them in a suitable vector representation. We extract a number of statistical features from each C-flow $c_i$, and translate them into $d$-dimensional pattern vectors $\vec{p_i} \in \mathbb{R}^d$. We can describe this task as a projection function $F$ : C-plane $\rightarrow \mathbb{R}^d$. The projection function $F$ is defined as follows. Given a C-flow $c_i$, we compute the discrete sample distribution of (currently) four random variables:

1. *the number of flows per hour (fph)*. *fph* is computed by counting the number of TCP/IP flows in $c_i$ that are present for each hour of the epoch $E$.
2. *the number of packets per flow (ppf)*. *ppf* is computed by summing the total number of packets sent within each TCP/UDP flow in $c_i$.
3. *the average number of bytes per packets (bpp)*. For each TCP/UDP flow $f_j \in c_i$ we divide the overall number of bytes transfered within $f_j$ by the number of packets sent within $f_j$.
4. *the average number of bytes per second (bps)*. *bps* is computed as the total number of bytes transfered within each $f_j \in c_i$ divided by the duration of $f_j$.

An example of the results of this process is shown in Figure 4, where we select a random client from a real network flow log (we consider a one-day epoch) and illustrate the features extracted from its visits to Google.

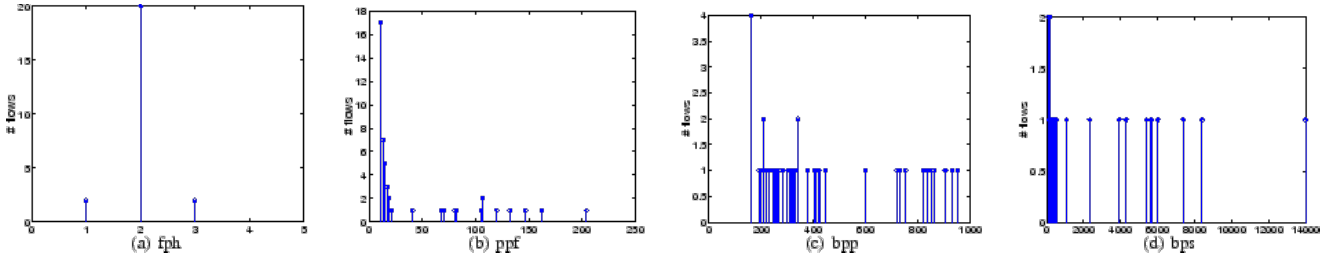


**Figure 4:** Visit pattern (shown in distribution) to Google from a randomly chosen normal client.

Given the discrete sample distribution of each of these four random variables, we compute an approximate version of it by means of a binning technique. For example, in order to approximate the distribution of *fph* we divide the x-axis in 13 intervals as $[0, k_1], (k_1, k_2], ..., (k_{12}, \infty)$. The values $k_1, .., k_{12}$ are computed as follows. First, we compute the overall discrete sample distribution of $fph$ considering all the C-flows in the traffic for an epoch $E$. Then, we compute the quantiles[7] $q_{5\%}, q_{10\%}, q_{15\%}, q_{20\%}, q_{25\%}, q_{30\%}, q_{40\%}, q_{50\%}, q_{60\%}, q_{70\%}, q_{80\%}, q_{90\%}$, of the obtained distribution, and we set $k_1 = q_{5\%}$, $k_2 = q_{10\%}$, $k_3 = q_{15\%}$, etc. Now, for each C-flow we can describe its *fph* (approximate) distribution as a vector of 13 elements, where each element $i$ represents the number of times *fph* assumed a value within the corresponding interval $(k_{i-1}, k_i]$. We also apply the same algorithm for *ppf*, *bpp*, and *bps*, and therefore we map each C-flow $c_i$ into a pattern vector $\vec{p_i}$ of $d = 52$ elements. Figure 5 shows the scaled visiting pattern extracted form the same C-flow

shown in Figure 4.

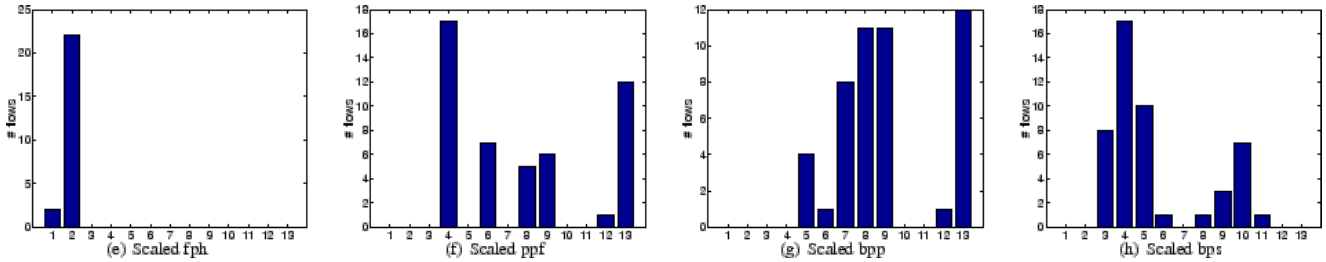


**Figure 5:** Scaled visit pattern (shown in distribution) to `Google` for the same client in Figure 4.

### 2.5.2 Two-step Clustering

Since bots belonging to the same botnet share similar behavior (from both the communication and activity points of view) as we discussed before, our objective is to look for groups of C-flows that are similar to each other. Intuitively, pattern vectors that are close to each other in $\mathbb{R}^d$ represent C-flows with similar communication patterns in the C-plane. For example, suppose two bots of the same botnet connect to two different C&C servers (because some botnets use multiple C&C servers). Although the connections from both bots to the C&C servers will be in different C-flows because of different source/destination pairs, their C&C traffic characteristics should be similar. That is, in $\mathbb{R}^d$, these C-flows should be found as being very similar. In order to find groups of hosts that share similar communication patterns, we apply clustering techniques on the dataset $\mathcal{D} = \{\vec{p}_i = F(c_i)\}_{i=1..n}$ of the pattern vector representations of C-flows. Clustering techniques perform unsupervised learning. Typically, they aim at finding meaningful groups of data points in a given feature space $\mathbb{F}$. The definition of ``meaningful clusters'' is application-dependent. Generally speaking, the goal is to group the data into clusters that are both compact and well separated from each other, according to a suitable similarity metric defined in the feature space $\mathbb{F}$ [25].

Clustering C-flows is a challenging task because $|\mathcal{D}|$, the cardinality of $\mathcal{D}$, is often large even for moderately large networks, and the dimensionality $d$ of the feature space is also large. Furthermore, because the percentage of machines in a network that are infected by bots is generally small, we need to separate the few botnet-related C-flows from a large number of benign C-flows. All these make clustering of C-flows very expensive.

In order to cope with the complexity of clustering of $\mathcal{D}$, we solve the problem in several steps (currently in two steps), as shown in a simple form in Figure 6.

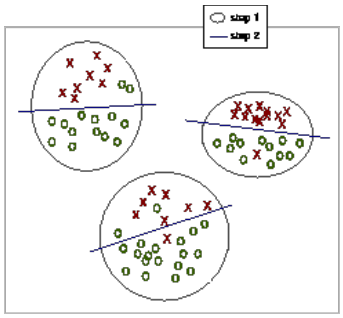


**Figure 6:** Two-step clustering of C-flows.

At the first step, we perform coarse-grained clustering on a reduced feature space $\mathbb{R}^{d'}$, with $d' < d$, using a simple (i.e., non-expensive) clustering algorithm (we will explain below how we perform dimensionality reduction). The results of this first-step clustering is a set $\{\mathcal{C}'_i\}_{i=1..\gamma_1}$ of $\gamma_1$ relatively large clusters. By doing so we subdivide the dataset $\mathcal{D}$ into smaller datasets (the clusters $\mathcal{C}'_i$) that contain ``clouds'' of points that are not too far from each other.

Afterwards, we refine this result by performing a second-step clustering on each different dataset $\mathcal{C}'_i$ using a simple clustering algorithm on the complete description of the C-flows in $\mathbb{R}^d$ (i.e., we do not perform dimensionality reduction in the second-step clustering). This second step generates a set of $\gamma_2$ smaller and more precise clusters $\{\mathcal{C}''_i\}_{i=1..\gamma_2}$.

We implement the first- and second-step clustering using the $X$-means clustering algorithm [31]. $X$-means is an efficient algorithm based on $K$-means [25], a very popular clustering algorithm. Different from $K$-means, the $X$-means algorithm does not require the user to choose the number $K$ of final clusters in advance. $X$-means runs multiple rounds of $K$-means internally and performs efficient clustering validation using the Bayesian Information Criterion [31] in order to compute the best value of $K$. $X$-means is fast and scales well with respect to the size of the dataset [31].

For the first-step (coarse-grained) clustering, we first reduce the dimensionality of the feature space from $d = 52$ features (see Section 2.5.1) into $d' = 8$ features by simply computing the mean and variance of the distribution of *fph*, *ppf*, *bpp*, and *bps* for each C-flow. Then we apply the $X$-means clustering algorithm on the obtained representation of C-flows to find the coarse-grained clusters $\{C'_i\}_{i=1..\gamma_1}$. Since the size of the clusters $\{C'_i\}_{i=1..\gamma_1}$ generated by the first-step clustering is relatively small, we can now afford to perform a more expensive analysis on each $C'_i$. Thus, for the second-step clustering, we use all the $d = 52$ available features to represent the C-flows, and we apply the $X$-means clustering algorithm to refine the results of the first-step clustering.

Of course, since unsupervised learning is a notoriously difficult task, the results of this two-step clustering algorithm may still be not perfect. As a consequence, the C-flows related to a botnet may be grouped into some distinct clusters, which basically represent *sub-botnets*. Furthermore, a cluster that contains mostly botnet or benign C-flows may also contain some ``noisy'' benign or botnet C-flows, respectively. However, we would like to stress the fact that these problems are not necessarily critical and can be alleviated by performing correlation with the results of the activity-plane (A-plane) clustering (see Section 2.7).

Finally, we need to note that it is possible to bootstrap the clustering from A-plane logs. For example, one may apply clustering to only those hosts that appear in the A-plane logs (i.e., the suspicious activity logs). This may greatly reduce the workload of the C-plane clustering module, if speed is the main concern. Similarly, one may bootstrap the A-plane correlation from C-plane logs, e.g., by monitoring only clients that previously formed communication clusters, or by giving monitoring preference to those clients that demonstrate some persistent C-flow communications (assuming botnets are used for long-term purpose).

## 2.6 A-plane Clustering

In this stage, we perform two-layer clustering on activity logs. Figure 7 shows the clustering process in A-plane.

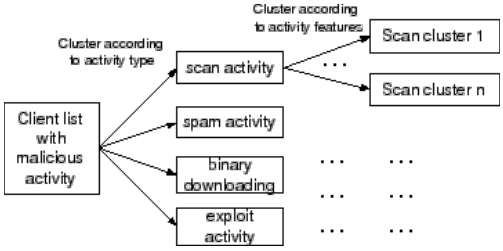


**Figure 7:** A-plane clustering.

For the whole list of clients that perform at least one malicious activity during one day, we first cluster them according to the types of their activities (e.g., scan, spam, and binary downloading). This is the first layer clustering. Then, for each activity type, we further cluster clients according to specific activity features (the second layer clustering). For scan activity, features could include scanning ports, that is, two clients could be clustered together if they are scanning the same ports. Another candidate feature could be the target subnet/distribution, e.g., whether the clients are scanning the same subnet. For spam activity, two clients could be clustered together if their SMTP connection destinations are highly overlapped. This might not be robust when the bots are configured to use different SMTP servers in order to evade detection. One can further consider the spam content if the whole SMTP traffic is captured. To cluster spam content, one may consider the similarity of embedded URLs that are very likely to be similar with the same botnet [43], SMTP connection frequency, content entropy, and the normalized compression distance (NCD [5,41]) on the entire email bodies. For outbound exploit activity, one can cluster two clients if they send the same type of exploit, indicated by the Snort alert SID. For binary downloading activity, two clients could be clustered together if they download similar binaries (because they download from the same URL as indicated in the command from the botmaster). A distance function between two binaries can be any string distance such as DICE used in [20] [8].

In our current implementation, we cluster scanning activities according to the destination scanning ports. For spam activity clustering, because there are very few hosts that show spamming activities in our monitored network, we

simply cluster hosts together if they perform spamming (i.e., using only the first layer clustering here). For binary downloading, we configure our binary downloading monitor to capture only the first portion (packet) of the binary for efficiency reasons (if necessary, we can also capture the entire binary). We simply compare whether these early portions of the binaries are the same or not. In other words, currently, our A-plane clustering implementation utilizes relatively weak cluster features. In the future, we plan to implement clustering on more complex feature sets discussed above, which are more robust against evasion. However, even with the current weak cluster features, BotMiner already demonstrated high accuracy with a low false positive rate as shown in our later experiments.

## 2.7 Cross-plane Correlation

Once we obtain the clustering results from A-plane (activities patterns) and C-plane (communication patterns), we perform cross-plane correlation. The idea is to cross-check clusters in the two planes to find out intersections that reinforce evidence of a host being part of a botnet. In order to do this, we first compute a botnet score $s(h)$ for each host $h$ on which we have witnessed at least one kind of suspicious activity. We filter out the hosts that have a score below a certain detection threshold $\theta$, and then group the remaining most suspicious hosts according to a similarity metric that takes into account the A-plane and C-plane clusters these hosts have in common.

We now explain how the botnet score is computed for each host. Let $H$ be the set of hosts reported in the output of the A-plane clustering module, and $h \in H$. Also, let $\mathcal{A}^{(h)} = \{A_i\}_{i=1..m_h}$ be the set of $m_h$ A-clusters that contain $h$, and $\mathcal{C}^{(h)} = \{C_i\}_{i=1..n_h}$ be the set of $n_h$ C-clusters that contain $h$. We compute the botnet score for $h$ as

$$s(h) = \sum_{\substack{i,j \\ j>i \\ t(A_i)\neq t(A_j)}} w(A_i)w(A_j)\frac{|A_i \cap A_j|}{|A_i \cup A_j|} + \sum_{i,k} w(A_i)\frac{|A_i \cap C_k|}{|A_i \cup C_k|}, \quad (1)$$

where $A_i, A_j \in \mathcal{A}^{(h)}$ and $C_k \in \mathcal{C}^{(h)}$, $t(A_i)$ is the type of activity cluster $A_i$ refers to (e.g., scanning or spamming), and $w(A_i) \geqslant 1$ is an *activity weight* assigned to $A_i$. $w(A_i)$ assigns higher values to ``strong'' activities (e.g., spam and exploit) and lower values to ``weak'' activities (e.g., scanning and binary download).

$h$ will receive a high score if it has performed multiple types of suspicious activities, and if other hosts that were clustered with $h$ also show the same multiple types of activities. For example, assume that $h$ performed scanning and then attempted to exploit a machine outside the monitored network. Let $A_1$ be the cluster of hosts that were found to perform scanning and were grouped with $h$ in the same cluster. Also, let $A_2$ be a cluster related to exploit activities that includes $h$ and other hosts that performed similar activities. A larger overlap between $A_1$ and $A_2$ would result in a higher score being assigned to $h$. Similarly, if $h$ belongs to A-clusters that have a large overlap with C-clusters, then it means that the hosts clustered together with $h$ share similar activities as well as similar communication patterns.

Given a predefined detection threshold $\theta$, we consider all the hosts $h \in H$ with $s(h) > \theta$ as (likely) bots, and filter out the hosts whose scores do not exceed $\theta$. Now, let $B \subseteq H$ be the set of detected bots, $\mathcal{A}^{(B)} = \{A_i\}_{i=1..m_B}$ be the set of A-clusters that each contains at least one bot $h \in B$, and $\mathcal{C}^{(B)} = \{C_i\}_{i=1..n_B}$ be the set of C-clusters that each contains at least one bot $h \in B$. Also, let $\mathcal{K}^{(B)} = \mathcal{A}^{(B)} \cup \mathcal{C}^{(B)} = \{K_i^{(B)}\}_{i=1..(m_B+n_B)}$ be an ordered union/set of A- and C-clusters. We then describe each bot $h \in B$ as a binary vector $b(h) \in \{0,1\}^{|\mathcal{K}^{(B)}|}$, whereby the $i$-th element $b_i = 1$ if $h \in K_i^{(B)}$, and $b_i = 0$ otherwise. Given this representation, we can define the following similarity between bots $h_i$ and $h_j$ as

$$sim(h_i, h_j) = \sum_{k=1}^{m_B} I(b_k^{(i)} = b_k^{(j)}) + I(\sum_{k=m_B+1}^{m_B+n_B} I(b_k^{(i)} = b_k^{(j)}) \geq 1), \quad (2)$$

where we use $b^{(i)} = b(h_i)$ and $b^{(j)} = b(h_j)$, for the sake of brevity. $I(X)$ is the indication function, which equals to one when the boolean argument $X$ is true, and equals to zero when $X$ is false. The intuition behind this metric is that if two hosts appear in the same activity clusters and in at least one common C-cluster, they should be clustered together.

This definition of similarity between hosts gives us the opportunity to apply hierarchical clustering. This allows us to build a dendrogram, i.e., a tree like graph (see Figure 8) that encodes the relationships among the bots. We use the Davies-Bouldin (DB) validation index [21] to find the best dendrogram cut, which produces the most compact and well separated clusters. The obtained clusters group bots in (sub-) botnets. Figure 8 shows a (hypothetical) example. Assuming that the best cut suggested by the DB index is the one at height 90, we would obtain two botnets, namely $\{h_8, h_3, h_5\}$, and $\{h_4, h_6, h_9, h_2, h_1, h_7\}$.

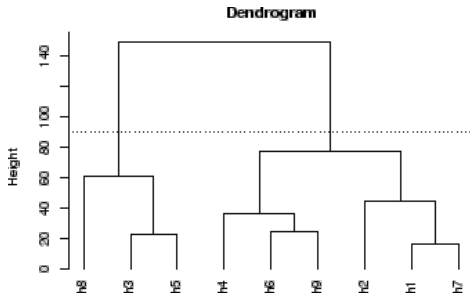


**Figure 8:** Example of hierarchical clustering for botnet detection.

In our current implementation, we simply set weight $w(A_i) = 1$ for all $i$ and $\theta = 0$, which essentially means that we will consider all hosts that appear in two different types of A-clusters and/or in both A- and C-clusters as suspicious candidates for further hierarchical clustering.

# 3 Experiments

To evaluate our BotMiner detection framework and prototype system, we have tested its performance on several real-world network traffic traces, including both (presumably) normal data from our campus network and collected botnet data.

## 3.1 Experiment Setup and Data Collection

We set up traffic monitors to work on a span port mirroring a backbone router at the campus network of the College of Computing at Georgia Tech. The traffic rate is typically 200Mbps-300Mbps at daytime. We ran the C-plane and A-plane monitors for a continuous 10-day period in late 2007. A random sampling of the network trace shows that the traffic is very diverse, containing many normal application protocols, such as `HTTP`, `SMTP`, `POP`, `FTP`, `SSH`, `NetBios`, `DNS`, `SNMP`, IM (e.g., `ICQ`, `AIM`), P2P (e.g., `Gnutella`, `Edonkey`, `bittorrent`), and `IRC`. This serves as a good background to test the false positives and detection performance on a normal network with rich application protocols.

We have collected a total of eight different botnets covering `IRC`, `HTTP` and P2P. Table 1 lists the basic information about these traces.

**Table 1:** Collected botnet traces, covering `IRC`, `HTTP` and P2P based botnets. Storm and Nugache share the same file, so the statistics of the whole file are reported.

| Trace | Size | Duration | Pkt | TCP/UDP flows | Botnet clients | C&C server |
|---|---|---|---|---|---|---|
| Botnet-IRC-rbot | 169MB | 24h | 1,175,083 | 180,988 | 4 | 1 |
| Botnet-IRC-sdbot | 66KB | 9m | 474 | 19 | 4 | 1 |
| Botnet-IRC-spybot | 15MB | 32m | 180,822 | 147,945 | 4 | 1 |
| Botnet-IRC-N | 6.4MB | 7m | 65,111 | 5635 | 259 | 1 |

| | | | | | | |
|---|---|---|---|---|---|---|
| Botnet-HTTP-1 | 6MB | 3.6h | 65,695 | 2,647 | 4 | 1 |
| Botnet-HTTP-2 | 37MB | 19h | 395,990 | 9,716 | 4 | 1 |
| Botnet-P2P-Storm | 1.2G | 24h | 59,322,490 | 5,495,223 | 13 | P2P |
| Botnet-P2P-Nugache | 1.2G | 24h | 59,322,490 | 5,495,223 | 82 | P2P |

We re-used two IRC and two HTTP botnet traces introduced in [20], i.e., V-Spybot, V-Sdbot, B-HTTP-I, and B-HTTP-II. In short, V-Spybot and V-Sdbot are generated by executing modified bot code (Spybot and Sdbot [6]) in a fully controlled virtual network. They contain four Windows XP/2K IRC bot clients, and last several minutes. B-HTTP-I and B-HTTP-II are generated based on the description of Web-based C&C communications in [24,39]. Four bot clients communicate with a controlled server and execute the received command (e.g., spam). In B-HTTP-I, the bot contacts the server periodically (about every five minutes) and the whole trace lasts for about 3.6 hours. In B-HTTP-II, we have a more stealthy C&C communication where the bot waits a random time between zero to ten minutes each time before it visits the server, and the whole trace lasts for 19 hours. These four traces are renamed as Botnet-IRC-spybot, Botnet-IRC-sdbot, Botnet-HTTP-1, and Botnet-HTTP-2, respectively. In addition, we also generated a new IRC botnet trace that lasts for a longer time (a whole day) using modified Rbot [3] source code. Again this is generated in a controlled virtual network with four Windows clients and one IRC server. This trace is labeled as Botnet-IRC-rbot.

We also obtained a real-world IRC-based botnet C&C trace that was captured in the wild in 2004, labeled as Botnet-IRC-N. The trace contains about 7-minute IRC C&C communications, and has hundreds of bots connected to the IRC C&C server. The botmaster set the command ``.scan.startall'' in the TOPIC of the channel. Thus, every bot would begin to propagate through scanning once joining the channel. They report their successful transfer of binary to some machines, and also report the machines that have been exploited. We believe this could be a variant of Phatbot [6]. Although we obtained only the IRC C&C traffic, we hypothesize that the scanning activities are easy to detect given the fact that bots are performing scanning commands in order to propagate. Thus, we assume we have an A-plane cluster with the botnet members because we want to see if we can still capture C-plane clusters and obtain cross-plane correlation results.

Finally, we obtained a real-world trace containing two P2P botnets, Nugache [28] and Storm [18,23]. The trace lasts for a whole day, and there are 82 Nugache bots and 13 Storm bots in the trace. It was captured from a group of honeypots running in the wild in late 2007. Each instance is running in Wine (an open source implementation of the Windows API on top of Unix/Linux) instead of a virtual or physical machine. Such a set-up is known as winobot [12] and is used by researchers to track botnets. By using a lightweight emulation environment (Wine), winobots can run hundreds and thousands of black-box instances of a given malware. This allows one to participate in a P2P botnet *en mass*. Nugache is a TCP-based P2P bot that performs encrypted communications on port 8. Storm, originating in January of 2007, is one of the very few known UDP based P2P bots. It is based on the Kademlia [30] protocol and makes use of the Overnet network [2] to locate related data (e.g., commands). Storm is well-known as a spam botnet with a huge number of infected hosts [27]. In the implementation of winobot, several malicious capabilities such as sending spam are disabled for legality reason, thus we can not observe spam traffic from the trace. However, we ran a full version of Storm on a VM-based honeypot (instead of Wine environment) and easily observed that it kept sending a huge amount of spam traffic, which makes the A-plane monitoring quite easy. Similarly, when running Nugache on a VM-based honeypot, we observed scanning activity to port 8 because it attempted to connect to its seeding peers but failed a lot of times (because the peers may not be available). Thus, we can detect and cluster A-plane activities for these P2P botnets.

## 3.2 Evaluation Results

**Table 2:** C-plane traffic statistics, basic results of filtering, and C-flows.

| Trace | Pkts | Flows | Filtered by F1 | Filtered by F2 | Filtered by F3 | Flows after filtering | C-flows (TCP/UDP) |
|---|---|---|---|---|---|---|---|
| Day-1 | 5,178,375,514 | 23,407,743 | 20,727,588 | 939,723 | 40,257 | 1,700,175 | 66,981 / 132,333 |
| Day-2 | 7,131,674,165 | 29,632,407 | 27,861,853 | 533,666 | 25,758 | 1,211,130 | 34,691 / 96,261 |
| Day-3 | 9,701,255,613 | 30,192,645 | 28,491,442 | 513,164 | 24,329 | 1,163,710 | 39,744 / 94,081 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Day-4 | 14,713,667,172 | 35,590,583 | 33,434,985 | 600,901 | 33,958 | 1,520,739 | 73,021 / 167,146 |
| Day-5 | 11,177,174,133 | 56,235,380 | 52,795,168 | 1,323,475 | 40,016 | 2,076,721 | 57,664 / 167,175 |
| Day-6 | 9,950,803,423 | 75,037,684 | 71,397,138 | 1,464,571 | 51,931 | 2,124,044 | 59,383 / 176,210 |
| Day-7 | 10,039,871,506 | 109,549,192 | 105,530,316 | 1,614,158 | 56,688 | 2,348,030 | 55,023 / 150,211 |
| Day-8 | 11,174,937,812 | 96,364,123 | 92,413,010 | 1,578,215 | 60,768 | 2,312,130 | 56,246 / 179,838 |
| Day-9 | 9,504,436,063 | 62,550,060 | 56,516,281 | 3,163,645 | 30,581 | 2,839,553 | 25,557 / 164,986 |
| Day-10 | 11,071,701,564 | 83,433,368 | 77,601,188 | 2,964,948 | 27,837 | 2,839,395 | 25,436 / 154,294 |

Table 2 lists the statistics for the 10 days of network data we used to validate our detection system. For each day there are around 5-10 billion packets (TCP and UDP) and 30-100 million flows. Table 2 shows the results of several steps of filtering. The first step of filtering (filter rule F1) seems to be the most effective filter in terms of data volume reduction. F1 filters out those flows that are not initiated from internal hosts to external hosts, and achieves about 90% data volume reduction. The is because most of the flows are within the campus network (i.e., they are initiated from internal hosts towards other internal hosts). F2 further filters out around 0.5-3 million of non-completely-established flows. F3 further reduces the data volume by filtering out another 30,000 flows. After applying all the three steps of filtering, there are around 1 to 3 million flows left per day. We converted these remaining flows into C-flows as described in Section 2.5, and obtained around 40,000 TCP C-flows and 130,000 UDP C-flows per day.

**Table 3:** C-plane and A-plane clustering results.

| Trace | Step-1 C-clusters | Step-2 C-clusters | A-plane logs | A-clusters | False Positive Clusters | FP Rate |
|---|---|---|---|---|---|---|
| Day-1 (TCP/UDP) | 1,374 | 4,958 | 1,671 | 1 | 0 | 0 (0/878) |
| Day-2 (TCP/UDP) | 904 | 2,897 | 5,434 | 1 | 1 | 0.003 (2/638) |
| Day-3 (TCP/UDP) | 1,128 | 2,480 | 4,324 | 1 | 1 | 0.003 (2/692) |
| Day-4 (TCP/UDP) | 1,528 | 4,089 | 5,483 | 4 | 4 | 0.01 (9/871) |
| Day-5 (TCP/UDP) | 1,051 | 3,377 | 6,461 | 5 | 2 | 0.0048 (4/838) |
| Day-6 (TCP) | 1,163 | 3,469 | 6,960 | 3 | 2 | 0.008 (7/877) |
| Day-7 (TCP) | 954 | 3,257 | 6,452 | 5 | 2 | 0.006 (5/835) |
| Day-8 (TCP) | 1,170 | 3,226 | 8,270 | 4 | 2 | 0.0091 (8/877) |
| Day-9 (TCP) | 742 | 1,763 | 7,687 | 2 | 0 | 0 (0/714) |
| Day-10 (TCP) | 712 | 1,673 | 7,524 | 0 | 0 | 0 (0/689) |

We then performed two-step clustering on C-flows as described in Section 2.5. Table 3 shows the clustering results and false positives (number of clusters that are not botnets). The results for the first 5 days are related to both TCP and UDP traffic, whereas in the last 5 days we focused on only TCP traffic.

It is easy to see from Table 3 that there are thousands of C-clusters generated each day. In addition, there are several thousand activity logs generated from A-plane monitors. Since we use relatively weak monitor modules, it is not surprising that we have this many activity logs. Many logs report binary downloading events or scanning activities. We cluster these activity logs according to their activity features. As explained early, we are interested in groups of machines that perform activities in a similar/coordinated way. Therefore, we filter out the A-clusters that contain only one host. This simple filtering rule allows us to obtain a small number of A-clusters and reduce the overall false positive rate of our botnet detection system.

**Table 4:** Botnet detection results using BotMiner.

| Botnet | Number of Bots | Detected? | Clustered Bots | Detection Rate | False Positive Clusters/Hosts | FP Rate |
|---|---|---|---|---|---|---|
| IRC-rbot | 4 | YES | 4 | 100% | 1/2 | 0.003 |
| IRC-sdbot | 4 | YES | 4 | 100% | 1/2 | 0.003 |
| IRC-spybot | 4 | YES | 3 | 75% | 1/2 | 0.003 |
| IRC-N | 259 | YES | 258 | 99.6% | 0 | 0 |
| HTTP-1 | 4 | YES | 4 | 100% | 1/2 | 0.003 |
| HTTP-2 | 4 | YES | 4 | 100% | 1/2 | 0.003 |
| P2P-Storm | 13 | YES | 13 | 100% | 0 | 0 |
| P2P-Nugache | 82 | YES | 82 | 100% | 0 | 0 |

Afterwards, we apply cross-plane correlation. We assume that the traffic we collected from our campus network is normal. In order to verify this assumption we used state-of-the-art botnet detection techniques like BotHunter [19] and BotSniffer [20]. Therefore, any cluster generated as a result of the cross-plane correlation is considered as a *false positive cluster*. It is easy to see from Table 3 that there are very few such false positive clusters every day (from zero to four). Most of these clusters contain only two clients (i.e., they induce two false positives). In three out of ten days no false positive was reported. In both Day-2 and Day-3, the cross-correlation produced one false positive cluster containing two hosts. Two false positive clusters were reported in each day from Day-5 to Day-8. In Day-4, the cross-plane correlation produced four false positive clusters.

For each day of traffic, the last column of Table 3 shows the false positive rate (FP rate), which is calculated as the fraction of IP addresses reported in the false positive clusters over the total number of distinct normal clients appearing in that day. After further analysis we found that many of these false positives are caused by clients performing binary downloading from websites not present in our whitelist. In practice, the number of false positives may be reduced by implementing a better binary downloading monitor and clustering module, e.g., by capturing the entire binary and performing content inspection (using either anomaly-based detection systems [38] or signature-based AV tools).

In order to validate the detection accuracy of BotMiner, we overlaid botnet traffic to normal traffic. We consider one botnet trace at a time and overlay it to the entire normal traffic trace of Day-2. We simulate a near-realistic scenario by constructing the test dataset as follows. Let $n$ be the number of distinct bots in the botnet trace we want to overlay to normal traffic. We randomly select $n$ distinct IP addresses from the normal traffic trace and map them to the $n$ IP addresses of the bots. That is, we replace an $IP_i$ of a normal machine with the $IP_i$ of a bot. In this way, we obtain a dataset of mixed normal and botnet traffic where a set of $n$ machines show both normal and botnet-related behavior. Table 4 reports the detection results for each botnet.

Table 4 shows that BotMiner is able to detect all eight botnets. We verified whether the members in the reported clusters are actually bots or not. For 6 out of 8 botnets, we obtained 100% detection rate, i.e., we successfully identified all the bots within the 6 botnets. For example, in the case of P2P botnets (`Botnet-P2P-Nugache` and `Botnet-P2P-Storm`), BotMiner correctly generated a cluster containing all the botnet members. In the case of `Botnet-IRC-spybot`, BotMiner correctly detected a cluster of bots. However, one of the bots belonging to the botnet was not reported in the cluster, which means that the detector generated a false negative. `Botnet-IRC-N` contains 259 bot clients. BotMiner was able to identify 258 of the bots in one cluster, whereas one of the bots was not detected. Therefore, in this case BotMiner had a detection rate of 99.6%.

There were some cases in which BotMiner also generated a false positive cluster containing two normal hosts. We verified that these two normal hosts in particular were also responsible for the false positives generated during the

analysis of the Day-2 normal traffic (see Table 3).

As we can see, BotMiner performs quite well in our experiments, showing a very high detection rate with relatively few false positives in real-world network traces.

# 4 Limitations and Potential Solutions

Like any intrusion/anomaly detection system, BotMiner is not perfect or complete. It is likely that once adversaries know our detection framework and implementation, they might find some ways to evade detection, e.g., by evading the C-plane and A-plane monitoring and clustering, or the cross-plane correlation analysis. We now address these limitations and discuss possible solutions.

## 4.1 Evading C-plane Monitoring and Clustering

Botnets may try to utilize a legitimate website (e.g., `Google`) for their C&C purpose in attempt to evade detection. Evasion would be successful in this case if we whitelisted such legitimate websites to reduce the volume of monitored traffic and improve the efficiency of our detection system. However, if a legitimate website, say `Google`, is used as a means to locate a secondary URL for actual command hosting or binary downloading, botnets may not be able to hide this secondary URL and the corresponding communications. Therefore, clustering of network traffic towards the server pointed by this secondary URL will likely allow us to detect the bots. Also, whitelisting is just an optional operation. One may easily choose not to use whitelisting to avoid such kind of evasion attempts (of course, in this case one may face the trade-off between accuracy and efficiency).

Botnet members may attempt to intentionally manipulate their communication patterns to evade our C-plane clustering. The easiest thing is to switch to multiple C&C servers. However, this does not help much to evade our detection because such peer communications could still be clustered together just like how we cluster P2P communications. A more advanced way is to randomize each individual communication pattern, for example by randomizing the number of packets per flow (e.g., by injecting random packets in a flow), and the number of bytes per packet (e.g., by padding random bytes in a packet). However, such randomization may introduce similarities among botnet members if we measure the distribution and entropy of communication features. Also, this randomization may raise suspicion because normal user communications may not have such randomized patterns. Advanced evasion may be attempted by bots that try to mimic the communication patterns of normal hosts, in a way similar to polymorphic blending attacks [15]. Furthermore, bots could use covert channes [1] to hide their actual C&C communications. We acknowledge that, generally speaking, communication randomization, mimicry attacks and covert channel represent limitations for all traffic-based detection approaches, including BotMiner's C-plane clustering technique. By incorporating more detection features such as content inspection and host level analysis, the detection system may make evasion more difficult.

Finally, we note that if botnets are used to perform multiple tasks (in A-plane), we may still detect them even when they can evade C-plane monitoring and analysis. By using the scoring algorithm described in Section 2.7, we can perform cross clustering analysis among multiple activity clusters (in A-plane) to accumulate the suspicious score needed to claim the existence of botnets. Thus, we may even *not* require C-plane analysis if there is already a *strong* cross-cluster correlation among different types of malicious activities in A-plane. For example, if the same set of hosts involve several types of A-plane clusters (e.g., they send spams, scan others, and/or download the same binaries), they can be reported as botnets because those behaviors, by themselves, are highly suspicious and most likely indicating botnets behaviors [19,20].

## 4.2 Evading A-plane Monitoring and Clustering

Malicious activities of botnets are unlikely or relatively hard to change as long as the botmaster wants the botnets to perform ``useful'' tasks. However, the botmaster can attempt to evade BotMiner's A-plane monitoring and clustering in several ways.

Botnets may perform very stealthy malicious activities in order to evade the detection of A-plane monitors. For example, they can scan very slowly (e.g., send one scan per hour), send spam very slowly (e.g., send one spam per day). This will evade our monitor sensors. However, this also puts a limit on the utility of bots.

In addition, as discussed above, if the botmaster commands each bot *randomly and individually* to perform different task, the bots are not different from previous generations of isolated, individual malware instances. This is unlikely the way a bot*net* is used in practice. A more advanced evasion is to differentiate the bots and avoid commanding bots in the same monitored network the same way. This will cause additional effort and inconvenience for the botmaster.

To defeat such an evasion, we can deploy distributed monitors on the Internet to cover a larger monitored space.

Note, if the botmaster takes the extreme action of randomizing/individualizing both the C&C communications and attack activities of each bots, then these bots are probably not part of a bot*net* according to our specific definition because the bots are not performing similar/coordinated commanded activities. Orthogonal to the *horizontal correlation* approaches such as BotMiner to detect a bot*net*, we can always use complementary systems like BotHunter [19] that examine the behavior history of *distinct* host for a dialog or *vertical correlation* based approach to detect *individual* bots.

### 4.3 Evading Cross-plane Analysis

A botmaster can command the bots to perform an extremely delayed task (e.g., delayed for days after receiving commands). Thus, the malicious activities and C&C communications are in different days. If only using one day's data, we may not be able to yield cross-plane clusters. As a solution, we may use multiple-day data and cross check back several days. Although this has the hope of capturing these botnets, it may also suffer from generating more false positives. Clearly, there is a trade-off. The botmaster also faces the trade-off because a very slow C&C essentially impedes the efficiency in controlling/coordinating the bot army. Also, a bot infected machine may be disconnected from the Internet or be powered off by the users during the delay and become unavailable to the botmaster.

In summary, while it is possible that a botmaster can find a way to exploit the limitations of BotMiner, the convenience or the efficiency of botnet C&C and the utility of the botnet also suffer. Thus, we believe that our protocol- and structure-independent detection framework represents a significant advance in botnet detection.

## 5 Related Work

To collect and analyze bots, researchers widely utilize honeypot techniques [4,32,16]. Freiling et al. [16] used honeypots to study the problem of botnets. Nepenthes [4] is a special honeypot tool for automatic malware sample collection. Rajab et al. [32] provided an in-depth measurement study of the current botnet activities by conducting a multi-faceted approach to collect bots and track botnets. Cooke et al. [10] conducted several basic studies of botnet dynamics. In [13], Dagon et al. proposed to use DNS sinkholing technique for botnet study and pointed out the global diurnal behavior of botnets. Barford and Yegneswaran [6] provided a detailed study on the code base of several common bot families. Collins et al. [9] presented their observation of a relationship between botnets and scanning/spamming activities.

Several recent papers proposed different approaches to detect botnets. Ramachandran et al.[34] proposed using DNSBL counter-intelligence to find botnet members that generate spams. This approach is useful for just certain types of spam botnets. In [35], Reiter and Yen proposed a system TAMD to detect malware (including botnets) by aggregating traffic that shares the same external destination, similar payload, and that involves internal hosts with similar OS platforms. TAMD's aggregation method based on destination networks focuses on networks that experience an increase in traffic as compared to a historical baseline. Different from BotMiner that focuses on botnet detection, TAMD aims to detect a broader range of malware. Since TAMD's aggregation features are different from BotMiner's (in which we cluster similar communication patterns and similar malicious activity patterns), TAMD and BotMiner can complement each other in botnet and malware detection. Livadas et al.[29,40] proposed a machine learning based approach for botnet detection using some general network-level traffic features of chat-like protocols such as IRC. Karasaridis et al.[26] studied network flow level detection of IRC botnet controllers for backbone networks. The above two are similar to our work in C-plane clustering but different in many ways. First, they are used to detect IRC-based botnet (by matching a known IRC traffic profile), while we do not have the assumption of known C&C protocol profiles. Second, we use a different feature set on a new communication flow (C-flow) data format instead of traditional network flow. Third, we consider both C-plane and A-plane information instead of just flow records.

Rishi [17] is a signature-based IRC botnet detection system by matching known IRC bot nickname patterns. Binkley and Singh [7] proposed combining IRC statistics and TCP work weight for the detection of IRC-based botnets. In [19], we described BotHunter, which is a passive bot detection system that uses dialog correlation to associate IDS events to a user-defined bot infection dialog model. Different from BotHunter's *dialog correlation* or *vertical correlation* that mainly examines the behavior history associated with each *distinct* host, BotMiner utilizes a *horizontal correlation* approach that examines correlation *across* multiple hosts. BotSniffer [20] is an anomaly-based botnet C&C detection system that also utilizes *horizontal correlation*. However, it is used mainly for detecting *centralized* C&C activities (e.g., IRC and HTTP).

The aforementioned systems are mostly limited to specific botnet protocols and structures, and many of them work only on IRC-based botnets. BotMiner is a novel general detection system that does not have such limitations and can greatly complement existing detection approaches.

# 6 Conclusion & Future Work

Botnet detection is a challenging problem. In this paper, we proposed a novel network anomaly-based botnet detection system that is independent of the protocol and structure used by botnets. Our system exploits the essential definition and properties of botnets, i.e., bots within the same botnet will exhibit similar C&C communication patterns and similar malicious activities patterns. In our experimental evaluation on many real-world network traces, BotMiner shows excellent detection accuracy on various types of botnets (including IRC-based, HTTP-based, and P2P-based botnets) with a very low false positive rate on normal traffic.

It is likely that future botnets (especially P2P botnets) may utilize evasion techniques to avoid detection, as discussed in Section 4. In our future work, we will study new techniques to monitor/cluster communication and activity patterns of botnets, and these techniques are intended to be more robust to evasion attempts. In addition, we plan to further improve the efficiency of the C-flow converting and clustering algorithms, combine different correlation techniques (e.g., vertical correlation and horizontal correlation), and develop new real-time detection systems based on a layered design using sampling techniques to work in very high speed and very large network environments.

# Acknowledgments

We would like to thank David Dagon and Yan Chen for their help in providing some of the evaluation data in our experiments. We thank Robert Edmonds for his support on using `fcapture`. In addition, we thank Angelos Stavrou, Rachna Dhamija, and anonymous reviewers for their insightful comments and feedback. This material is based upon work supported in part by the National Science Foundation under Grants CCR-0133629, CNS-0627477, and CNS-0716570, by the U.S. Army Research Office under Grant W911NF0610042, and by the Air Force Research Laboratory (AFRL) under Grant FA8750-08-2-0141. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation, the U.S. Army Research Office, and the Air Force Research Laboratory.

## Bibliography

1

A guide to understanding covert channel analysis of trusted systems, version 1.
NCSC-TG-030, Library No. S-240,572, National Computer Security Center, November 1993.

2

Overnet.
http://en.wikipedia.org/wiki/Overnet, 2008.

3

P. Bacher, T. Holz, M. Kotter, and G. Wicherski.
Know your enemy: Tracking botnets.
http://www.honeynet.org/papers/bots/, 2005.

4

P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. Freiling.
The nepenthes platform: An efficient approach to collect malware.
In *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID'06)*, Hamburg, September 2006.

5

M. Bailey, J. Oberheide, J. Andersen, M. Mao, F. Jahanian, and J. Nazario.
Automated classification and analysis of internet malware.
In *Proceedings of International Symposium on Recent Advances in Intrusion Detection (RAID'07)*, 2007.

6

P. Barford and V. Yegneswaran.
An Inside Look at Botnets.

Special Workshop on Malware Detection, Advances in Information Security, Springer Verlag, 2006.

7
J. R. Binkley and S. Singh.
An algorithm for anomaly-based botnet detection.
In *Proceedings of USENIX SRUTI'06*, pages 43-48, July 2006.

8
K. Chiang and L. Lloyd.
A case study of the rustock rootkit and spam bot.
In *Proceedings of USENIX HotBots'07*, 2007.

9
M. Collins, T. Shimeall, S. Faber, J. Janies, R. Weaver, M. D. Shon, and J. Kadane.
Using uncleanliness to predict future botnet addresses,.
In *Proceedings of ACM/USENIX Internet Measurement Conference (IMC'07)*, 2007.

10
E. Cooke, F. Jahanian, and D. McPherson.
The zombie roundup: Understanding, detecting, and disrupting botnets.
In *Proceedings of USENIX SRUTI'05*, 2005.

11
Cyber-TA.
BotHunter Free Internet Distribution Page.
http://www.cyber-ta.org/BotHunter, 2008.

12
D. Dagon, G. Gu, C. Lee, and W. Lee.
A taxonomy of botnet structures.
In *Proceedings of the 23 Annual Computer Security Applications Conference (ACSAC'07)*, 2007.

13
D. Dagon, C. Zou, and W. Lee.
Modeling botnet propagation using timezones.
In *Proceedings of the 13th Annual Network and Distributed System Security Symposium (NDSS'06)*, January 2006.

14
N. Daswani and M. Stoppelman.
The anatomy of clickbot.a.
In *Proceedings of USENIX HotBots'07*, 2007.

15
P. Fogla, M. Sharif, R. Perdisci, O. M. Kolesnikov, and W. Lee.
Polymorphic blending attack.
In *Proceedings of the 15th USENIX Security Symposium (Security'06)*, 2006.

16
F. Freiling, T. Holz, and G. Wicherski.
Botnet Tracking: Exploring a Root-cause Methodology to Prevent Denial of Service Attacks.
In *Proceedings of 10th European Symposium on Research in Computer Security (ESORICS'05)*, 2005.

17
J. Goebel and T. Holz.
Rishi: Identify bot contaminated hosts by irc nickname evaluation.
In *Proceedings of USENIX HotBots'07*, 2007.

18
J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon.
Peer-to-peer botnets: Overview and case study.
In *Proceedings of USENIX HotBots'07*, 2007.

19
G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee.
BotHunter: Detecting malware infection through ids-driven dialog correlation.

In *Proceedings of the 16th USENIX Security Symposium (Security'07)*, 2007.

20
G. Gu, J. Zhang, and W. Lee.
BotSniffer: Detecting botnet command and control channels in network traffic.
In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, 2008.

21
M. Halkidi, Y. Batistakis, and M. Vazirgiannis.
On clustering validation techniques.
*J. Intell. Inf. Syst.*, 17(2-3):107-145, 2001.

22
T. Holz, C. Gorecki, K. Rieck, and F. C. Freiling.
Detection and mitigation of fast-flux service networks.
In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, 2008.

23
T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling.
Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm.
In *Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'08)*, 2008.

24
N. Ianelli and A. Hackworth.
Botnets as a vehicle for online crime.
http://www.cert.org/archive/pdf/Botnets.pdf, 2005.

25
A. K. Jain, M. N. Murty, and P. J. Flynn.
Data clustering: a review.
*ACM Computer Survey*, 31(3):264-323, 1999.

26
A. Karasaridis, B. Rexroad, and D. Hoeflin.
Wide-scale botnet detection and characterization.
In *Proceedings of USENIX HotBots'07*, 2007.

27
B. Krebs.
Storm worm dwarfs world's top supercomputers.
http://blog.washingtonpost.com/securityfix/2007/08/storm_worm_dwarfs_worlds_top_s_1.html, 2007.

28
R. Lemos.
Bot software looks to improve peerage.
Http://www.securityfocus.com/news/11390, 2006.

29
C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer.
Using machine learning techniques to identify botnet traffic.
In *Proceedings of the 2nd IEEE LCN Workshop on Network Security (WoNS'2006)*, 2006.

30
P. Maymounkov and D. Mazieres.
Kademlia: A peer-to-peer information system based on the XOR metric.
In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS'02)*, 2002.

31
D. Pelleg and A. W. Moore.
X-means: Extending k-means with efficient estimation of the number of clusters.
In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML'00)*, pages 727-734, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

32
M. Rajab, J. Zarfoss, F. Monrose, and A. Terzis.
A multi-faceted approach to understanding the botnet phenomenon.

In *Proceedings of ACM SIGCOMM/USENIX Internet Measurement Conference (IMC'06)*, Brazil, October 2006.

33
A. Ramachandran and N. Feamster.
Understanding the network-level behavior of spammers.
In *Proceedings of ACM SIGCOMM'06*, 2006.

34
A. Ramachandran, N. Feamster, and D. Dagon.
Revealing botnet membership using DNSBL counter-intelligence.
In *Proceedings of USENIX SRUTI'06*, 2006.

35
M. K. Reiter and T.-F. Yen.
Traffic aggregation for malware detection.
In *Proceedings of the Fifth GI International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'08)*, 2008.

36
M. Roesch.
Snort - lightweight intrusion detection for networks.
In *Proceedings of USENIX LISA'99*, 1999.

37
P. Royal, M. Halpin, D. Dagon, R. Edmonds, and W. Lee.
Polyunpack: Automating the hidden-code extraction of unpack-executing malware.
In *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC'06)*, pages 289-300, Washington, DC, USA, 2006. IEEE Computer Society.

38
M. G. Schultz, E. Eskin, E. Zadok, and S. J. Stolfo.
Data mining methods for detection of new malicious executables.
In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, 2001.

39
SecureWorks.
Bobax trojan analysis.
http://www.secureworks.com/research/threats/bobax/, 2004.

40
W. T. Strayer, R. Walsh, C. Livadas, and D. Lapsley.
Detecting botnets with tight command and control.
In *Proceedings of the 31st IEEE Conference on Local Computer Networks (LCN'06)*, 2006.

41
S. Wehner.
Analyzing worms and network traffic using compression.
*Journal of Computer Security*, 15(3):303-320, 2007.

42
T. Werner.
PE Hunter.
http://honeytrap.mwcollect.org/pehunter, 2007.

43
L. Zhuang, J. Dunagan, D. R. Simon, H. J. Wang, I. Osipkov, G. Hulten, and J. Tygar.
Characterizing botnets from email spam records.
In *Proceedings of the First USENIX Workshop on Large-Scale Exploits and Emergent Threats (LEET'08)*, 2008.

44
J. Zhuge, T. Holz, X. Han, J. Guo, and W. Zou.
Characterizing the irc-based botnet phenomenon.
Peking University & University of Mannheim Technical Report, 2007.

**Footnotes**

...HTTP[1]
BotSniffer could be extended to support other protocol based C&C, if the corresponding protocol matchers are added.

... DDoS[2]
For spreading, the scans usually span many different hosts (within a subnet) indicated by the botnet command. For DDoS, usually there are numerous connection attempts to a specific host. In both cases, the traffic can be considered as scanning related.

... work[3]
One can still use our complementary system, BotHunter [19], to detect individual bots. In this paper, we focus on the detection of a bot*net*. We further clarify our assumptions in Section 2.1 and address limitations in Section 4.

...BotSniffer2008[4]
All these tools can also be deployed in LANs.

...fcapture [5]
This tool will be released in open source soon.

... hosts[6]
If the C-plane monitor is deployed at the edge router, these traffic will not be seen. However, if the monitor is deployed/tested in a LAN, then this filtering can be used.

... quantiles[7]
The quantile $q_{l\%}$ of a random variable $X$ is the value $q$ for which $P(X < q) = l\%$.

...BotSniffer2008 [8]
In an extreme case that bots update their binaries from different URLs (and the binaries are packed to be polymorphic thus different from each other), one should unpack the binary using tools such as Polyunpack [37] before calculating the distance. One may also directly apply normalized compression distance (NCD [5,41]) on the original (maybe packed) binaries.

---

*Guofei Gu 2008-05-09*