Detecting Stealthy P2P Botnets Using Statistical Traffic Fingerprints

Junjie Zhang†, Roberto Perdisci‡, Wenke Lee†, Unum Sarfraz† and Xiapu Luo*
†Georgia Institute of Technology, ‡University of Georgia, *Hong Kong Polytechnic University
{jjzhang,wenke}@cc.gatech.edu, unum.sarfraz@gatech.edu
perdisci@cs.uga.edu,csxluo@comp.polyu.edu.hk

***Abstract*—Peer-to-peer (P2P) botnets have recently been adopted by botmasters for their resiliency to take-down efforts. Besides being harder to take down, modern botnets tend to be stealthier in the way they perform malicious activities, making current detection approaches, including [6], ineffective. In this paper, we propose a novel botnet detection system that is able to identify stealthy P2P botnets, even when malicious activities may not be observable. First, our system identifies all hosts that are likely engaged in P2P communications. Then, we derive statistical fingerprints to profile different types of P2P traffic, and we leverage these fingerprints to distinguish between P2P botnet traffic and other legitimate P2P traffic. Unlike previous work, our system is able to detect stealthy P2P botnets even when the underlying compromised hosts are running legitimate P2P applications (e.g., Skype) and the P2P bot software at the same time. Our experimental evaluation based on real-world data shows that the proposed system can achieve high detection accuracy with a low false positive rate.**

***Keywords*-Botnet, P2P, Intrusion Detection, Security.**

I. Introduction

A botnet is a collection of compromised hosts (a.k.a. bots) that are remotely controlled by an attacker (the botmaster) via a command and control (C&C) channel. Botnets serve as the infrastructures for a variety of cyber-crimes, such as sending spam, launching distributed denial-of-service (DDoS) attacks, performing identity theft, click fraud, etc.

The C&C channel is an essential component of a botnet. Botmasters rely on the C&C channel to issue commands to their bots and receive information from the compromised machines. Different botnets may structure their C&C channel in different ways. In a centralized architecture, all the bots in a botnet contact one (or a few) C&C server(s) *owned* by the botmaster. Centralized C&C channels based on the `IRC` or `http` protocol have been used by many botnets due to their simplicity and availability of open-source, reusable C&C server code. However, centralized C&C servers represent a *single point of failure*. Therefore, attackers have recently started to build botnets with a more resilient C&C architecture, using a peer-to-peer (P2P) structure [13, 14, 18] or hybrid P2P/centralized C&C structures [17]. Bots belonging to a P2P botnet (i.e., a botnet that uses P2P-based C&C communications) form an overlay network in which any of the nodes (i.e., any of the bots) can be used by the botmaster to distribute commands to the other peers or collect information from them. While more complex, and perhaps more costly to manage compared to centralized botnets, P2P botnets offer higher resiliency, since even if a significant portion of a P2P botnet is taken down (e.g., by law enforcement or network operators) the remaining bots may still be able to communicate with each other and with the botmaster. Notable examples of P2P botnets are represented by `Nugache` [10], `Storm` [13], `Waledac` [17], and even `Confiker`, which has been shown to embed P2P capabilities [14]. `Storm` and `Waledac` are of particular interest because they use P2P C&C structures as the primary way to organize their bots, and have demonstrated resilience to take-down attempts[1].

To date, a few approaches for P2P botnet detection have been proposed [6, 15, 19]. BotMiner [6] finds groups of hosts within a monitored network that share similar communication patterns with outside machines and at the same time perform similar malicious activities, such as scanning, spamming, launching remote exploits, etc. If such groups of hosts exist, they are considered to be part of a botnet and an alarm is raised. The intuition is that bots belonging to the same botnet will share similar C&C communication patterns, and will respond to the botmaster's commands with similar malicious activities. Unfortunately, modern botnets are using more and more stealthy ways to perform malicious activities. For example, some botnets may send spam through large popular webmail services such as `Gmail` or `Hotmail` [22]. Such activities are very hard to detect through network flow analysis, due to encryption and overlap with legitimate webmail usage patterns, thus making BotMiner ineffective. BotGrep [15] is based on analysis of network flows collected over multiple large networks (e.g., ISP networks), and attempts to detect P2P botnets by analyzing the communication graph formed by overlay networks. Starting from a global view of Internet traffic, BotGrep first identifies groups of hosts that form a P2P network. To further differentiate P2P botnets from the legitimate P2P networks (e.g., P2P file sharing networks), BotGrep requires additional information to bootstrap its detection algorithm. For example, BotGrep may use a list of nodes in a communication (sub-)graph that are related to *honeypot* hosts, or may leverage the detection results from intrusion detection systems. However, acquiring both a sufficiently global view of Internet communications and enough *a priori* information to bootstrap the detection algorithm may be very challenging and makes the detection results (which in [22] were mainly based on simulations)

[1]After extensive effort, both `Storm` and `Waledac` have been recently taken down by network operators.

heavily dependent on other systems, thus limiting the real-world applicability of BotGrep. Recently, Yen et al. [19] have proposed an algorithm that aims to distinguish between hosts that run legitimate P2P file sharing applications and P2P bots. However, the proposed algorithm [19] does not take into account the fact that some popular legitimate P2P applications may not exhibit network patterns typical of P2P file sharing applications. For example, `Skype`, a very popular P2P-based instant messenger, does not usually behave in a way similar to file sharing applications. For example, large file transfers through Skype are usually rare, compared to its use as instant messenger or voice-over-IP (VoIP) client. Therefore, Skype's P2P traffic may cause a significant number of false positives. Moreover, the algorithm in [19] is not able to detect bot-compromised hosts that exhibit mixed legitimate and botnet-related P2P traffic (e.g., due to users running a file sharing P2P application on machines compromised with P2P bots).

In this paper, we present a novel botnet detection system that is able to identify *stealthy* P2P botnets. Our system aims to detect all P2P botnets, even in the case in which their malicious activities may not be observable. The approach we propose focuses on identifying P2P bots within a monitored network by detecting the C&C communication patterns that characterize P2P botnets, regardless of how they perform malicious activities in response to the botmaster's commands. To accomplish this task, we first identify all hosts within a monitored network that appear to be engaging in P2P communications. Then, we derive *statistical fingerprints* of the P2P communications generated by these hosts, and leverage the obtained fingerprints to distinguish between hosts that are part of legitimate P2P networks (e.g., file-sharing networks) and P2P bots. Unlike previous work, our system is able to identify stealthy P2P bots within a monitored network even when the P2P botnet traffic is overlapped with traffic generated by legitimate P2P applications (e.g., Skype) running on the same compromised host.

Our work makes the following contributions:

1) A new *flow-clustering*-based analysis approach to identify hosts that are most likely running P2P applications, and estimate the *active time* of the detected P2P nodes.
2) An efficient algorithm for P2P traffic *fingerprinting*, which we use to build a statistical profile of different P2P applications.
3) A P2P botnet detection system that can effectively and accurately detect P2P bots, even in the case in which they perform malicious activities in a stealthy, non-observable way. In addition, our system is able to identify bot-compromised machines, even in the case in which the P2P botnet traffic is overlapped with traffic generated by legitimate P2P applications (e.g., Skype) running on the same compromised machine.
4) An implementation of our detection system, and an extensive experimental evaluation. Our experimental results show that we can detect P2P bots with a detection rate of 100% and 0.2% false positive rate.

II. Related Work

As P2P botnets become robust infrastructures for various malicious activities, they have attracted a lot of effort from researchers [8, 9, 13, 14, 18]; the most notable and studied P2P botnets are `Nugache` [18], `Storm` [8, 13], `Waledac` [17] and `Confiker` [14]. A few approaches have been proposed that can be used for P2P botnet detection [6, 15, 19], which have been discussed in Section I. `BotHunter` [5] was proposed to detect a bot, centralized or P2P, in its *infection phase* if infection behaviors are consistent with the infection model used by `BotHunter`. However, bots now use a wide variety of approaches for infection (e.g., drive-by downloads), which may not be consistent with `BotHunter`'s infection model.

Our work focuses on the detection of P2P botnets using network information. Compared with the existing approaches, the design goals of our approach are different in that: 1) our approach does not need to assume that malicious activities are observable, unlike [6]; 2) our approach does not require any botnet-specific information to make the detection, unlike [15]; and 3) our approach needs to detect the compromised hosts that run both P2P bot and other legitimate P2P applications at the same time, unlike [19]. To achieve these design goals, our system includes multiple components. The first one is a *flow-clustering*-based analysis approach to identify hosts that are mostly likely running P2P applications. In contrast to existing approaches of identifying hosts running P2P applications [3, 12, 16, 20, 21], our approach differs from them in the following ways: 1) unlike [16], our approach does not need any content signature because encryption will immediately make content signature useless; 2) our approach does not rely on any transport layer heuristics (e.g., fixed source port) used by [20, 21], which can be easily violated by P2P applications; 3) we do not need training data set to build a machine learning based model as used in [3], because it is very challenging to get traffic of P2P botnets before they are detected; 4) in contrast to [12], our approach can detect and profile various P2P applications rather than identifying a specific P2P application (e.g., `Bittorrent`); and 5) our analysis approach can estimate the active time of a P2P application, which is critical for botnet detection.

III. Detection System

Problem Formulation: Our goal is to monitor the network traffic at the edge of a network (e.g., an enterprise network), and identify whether any of the machines within the network perimeter has become part of a P2P botnet. In particular, we consider the scenario in which bots perform malicious activities in a stealthy way, for example spam-

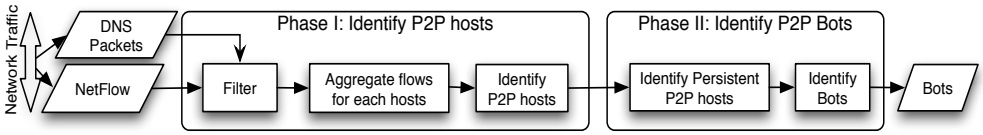


Figure 1: System Overview

bots that send spam through stolen or malicious web-mail accounts (e.g., `Gmail` or `Hotmail` accounts) [22], whose malicious activities are very hard to detect from traffic analysis. In general, we assume that the bots' malicious activities may not be easily observable, and therefore we only focus on their C&C communication patterns. We assume that at least two or more machines within the monitored network are part of the same P2P botnet, and leverage the similarity in communication patterns across multiple bots for detection purposes.

System Overview: P2P-based botnets rely on a P2P protocol to establish a C&C channel and communicate with the botmaster. As such, we intuitively assume that P2P bots exhibit some network traffic patterns that are common to other P2P client applications (either legitimate or malicious). Therefore, we divide our systems into two phases. As the first phase, we aim at detecting all hosts within the monitored network that appear to be engaging in P2P communications, as shown in Figure 1. We analyze raw traffic collected at the edge of the monitored network (e.g., an enterprise network), and apply a pre-filtering step (discussed in Section III-A) to reduce the data volume and only consider network flows that are potentially related to P2P communications. Then, we analyze the remaining traffic and extract a number of statistical features (described in Section III-B), which we use to isolate flows related to P2P communications from unrelated flows, and identify *candidate* P2P clients.

In the second phase, our botnet detection system (detailed in Section III-D) analyzes the traffic generated by the candidate P2P clients and classifies them into either *legitimate* P2P clients or P2P *bots*. The architecture of our botnet detection system is based on a number of observations. First, bots are malicious programs used to perform profitable malicious activities. They represent valuable assets for the botmaster, who will intuitively try to maximize their *utilization*. As a consequence, bot programs usually make themselves persistent on the compromised system and run for as long as the system is powered on. This is particularly true for P2P bots, because in order to have a functional overlay network (the botnet), a sufficient number of peers needs to be always online. In other words, the active time of a bot should be comparable with the active time of the underlying compromised system. If this was not the case, the botnet overlay network would risk to *degenerate* into a number of disconnected subnetworks, due to the short life time of each single node.

On the other hand, the active time of legitimate P2P applications is determined by users. For example, some users tend to use their file-sharing P2P clients only to download a limited number of files, before shutting down the P2P application [4]. In this case, the active time of the legitimate P2P application may be much shorter compared to the active time of the underlying system. Based on this observation, our botnet detection system first estimates the active time of a P2P client and eliminates those hosts that are running P2P applications with short active time, compared to the underlying system. It is worth noting that some users may run certain legitimate P2P applications for as long as their machine is on. For example, `Skype` is a popular P2P application for instant messaging and voice-over-IP (VoIP) that is often setup to start after system boot, and that keeps running until the system is turned off. Therefore, such `Skype` clients (or other "persistent" P2P clients) will not be filtered out at this stage.

In order to discriminate between legitimate *persistent* P2P clients and P2P bots, we make use of the following observations: 1) bots that belong to the same botnet use the same P2P protocol and network, and 2) the set of peers contacted by two different bots have a much larger overlap, compared to peers contacted by two P2P clients connected to the same legitimate P2P network. While the first observation is obvious, the second observation deserves explanation.

Assume that two hosts in the monitored network, say h_A and h_B, are running the same legitimate P2P file-sharing application (e.g., Emule). The users of these two P2P clients will most likely have uncorrelated usage patterns. Namely, it is reasonable to assume that in the general case the two users will search for and download different content (e.g., different media files or documents) from the P2P network. This translates into a *divergence* between the set of IP addresses contacted by hosts h_A and h_B (remember that at this stage we are only considering the P2P traffic generated by the hosts). The reason is that the two P2P clients will tend to exchange P2P control messages (e.g., ping/pong and search requests) with different sets of peers which "own" the content requested by their users, or peers that are *along the path* towards the content. On the contrary, assume that hosts h_A and h_B are compromised with P2P bots. One of the characteristics of the bots is that they need to periodically *search for commands* published by the botmaster [8]. This typically translates into a *convergence* between the set of IPs contacted by h_A and h_B (we will discuss potential exceptions to this behavior in Section V).

To summarize, in order to detect P2P bots we follow the

P2P Apps	Version	Protocol
Bittorrent	6.4	Bittorrent
Emule	0.49c	Kademlia
Limewire	5.4.8	Gnutella&Bittorrent
Skype	4.2	Skype
Ares	2.1.5	Gnutella&Bittorrent

Table I: P2P Applications

notation	Description
T_{p2p}	the active time of P2P application
N_f	the number of failed connections per hour
No-DNS Peers	the percentage of flows associated with no domain names
N_{clust}	the number of clusters left by enforcing Θ_{bgp} and Θ_{p2p}
N_{bgp}	the largest number of unique bgp prefixes in one cluster
$\hat{T_{p2p}}$	the estimated active time for P2P application

Table II: Notations and Descriptions

high-level steps reported below:

1) Identify the set $\mathbf{H}$ of all hosts engaged in P2P communications.
2) Identify the subset $\mathbf{P} \subseteq \mathbf{H}$ of P2P clients whose active time is similar to the active time of the underlying systems.
3) Identify the subset $\mathbf{B} \subseteq \mathbf{P}$ which exhibit similar P2P communication patterns, and a significant overlap of the set of contacted peers. We classify the hosts in set $\mathbf{B}$ as P2P bots.

To illustrate the statistical features and motivate the related thresholds used by our system, we used five popular P2P applications (see Table I) for 24 hours to collect their traffic traces. For the Bittorrent application, we generated two separate 24-hour traces (`T-Bittorrent` and `T-Bittorrent-2`). In this section we report a number of measurements on the obtained traffic traces to better motivate some of our design choices. Table III reports the feature values measured on the collected traffic traces. The notation used for our statistical features is summarized in Table II.

We now describe the components of our detection system in more details.

A. Traffic Volume Reduction

As a first step, we want to filter out network traffic (and their sources) that is unlikely to be related to P2P communications. This is accomplished in part by passively analyzing DNS traffic, and identifying network flows whose destination IP address was previously *resolved* in a DNS response. The reason is that P2P clients usually contact their peers directly, by looking up IPs from a routing table for the overlay network, rather than resolving a domain name (a possible exception may be when a peer bootstraps into a P2P network by looking up domain names that resolve to stable *super-nodes*). This observation is supported by Table III (*No-DNS Peers*), which illustrates the percentage of flows whose destination IP was not resolved from a domain name. It confirms that the vast majority of flows generated by P2P applications do not have destination IPs resolved from domain names. The remaining small fraction of flows are either related to bootstrapping (e.g., in the case of bittorrent.com and skype.com) or for downloading advertisement content from popular websites. Since most non-P2P applications (e.g., browsers, email clients, etc.) often connect to a destination address resulting from domain name resolution, this simple filter can eliminate a very large percentage of non-P2P traffic (see Section IV) while retaining the vast majority of P2P communications.

Trace	T_{p2p}	N_f	*No-DNS Peers*	N_{clust}	N_{bgp}	$\hat{T_{p2p}}$
T-Bittorrent	24hr	1602	96.85%	17	12857	24hr
T-Emule	24hr	318	99.99%	8	1133	24hr
T-Limewire	24hr	1278	99.97%	36	5661	24hr
T-Skype	24hr	81	99.93%	12	12806	24hr
T-Ares	24hr	489	99.99%	16	1596	24hr

Table III: Measurement of Features

B. Identifying P2P Clients

After traffic volume reduction we consider the remaining traffic, and for each host h within the monitored network we identify three flow sets (we call "outgoing" those flows that have been initiated by h):

1) $S_{tcp}(h)$: flows related to successful outgoing TCP connections.
2) $S_{udp}(h)$: flows related to successful outgoing UDP (virtual) connections.
3) $S_o(h)$: flows related to failed outgoing TCP/UDP connections.

We consider as successful those TCP connections with a completed `SYN, SYN/ACK, ACK` handshake, and those UDP (virtual) connections for which there was at least one "request" packet and a consequent response packet.

P2P applications act as both clients and servers. A node in a P2P network can initiate (TCP or UDP virtual) connections to its peers and accept connections initiated by other peers. In client-mode, P2P nodes periodically probe their peers with `ping/pong` messages to maintain a view of the overlay network (usually for routing purposes), or search for content. A consequence of this behavior is the fact that P2P nodes will often generate a large number of failed outgoing flows. The reason is that P2P networks are usually characterized by a significant node churn [4], due to previous nodes that leave the network and new nodes that join it (the churn is intuitively correlated with users that turn on or off their P2P applications or machines). Therefore, a node that sends a ping message to a known peer will often discover that the peer is not up anymore (no pong is received, thus causing a failed connection).

At this point, we retain all hosts that generated at least a successful outgoing TCP or UDP connection, and that generated more than a predefined number Θ_o of outgoing failed TCP/UDP connections. Namely, we retain a host h if $|S_{tcp}(h)| + |S_{udp}(h)| > 0$ AND $S_o(h) > \Theta_o$, and discard all other hosts (it is worth noting that here we are only considering those flows that passed the DNS-based traffic volume reduction filter described in Section III-A). Table III, reports the number N_f of failed outgoing connections per

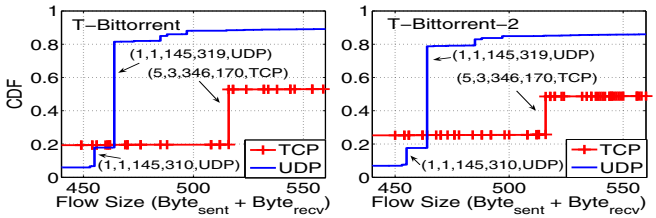


Figure 2: CDF of Flow Size

hour for different P2P applications. We can see that P2P applications typically generate a large number (from several tens up to thousands) of failed connection attempts with other peers. Therefore, we conservatively set $\Theta_o = 10$.

What we just described is a coarse-grained filter that allows us to focus on *candidate* P2P nodes. We further apply a more fine-grained analysis to prune away hosts that are not actual P2P nodes. For example, we want to eliminate hosts that made it into the list of *candidate* P2P nodes by chance (e.g., because of scanning behavior). To this end, we first consider the fact that each node of a P2P network frequently exchanges a number of control messages (e.g., ping/pong messages) with other peers. Also, we notice that the characteristics of these messages, such as the size and frequency of the exchanged packets, are *similar* for nodes in the same P2P network, and *vary* depending on the P2P protocol and network in use. In addition, we notice that a node will often exchange control messages with a relatively large number of peers distributed in many different networks, where each network can be represented by its `BGP` prefix. Figure 2 describes the distribution of flow sizes for two `Bittorrent` traces, where a large number of flows share similar sizes.

To identify flows corresponding to P2P control messages, we first apply a flow clustering process intended to group together similar flows for each candidate P2P node h. Given sets of flows $S_{tcp}(h)$ and $S_{udp}(h)$, we describe each flow as a vector of statistical features $v(h) = [Pkt_s, Pkt_r, Byte_s, Byte_r]$, in which Pkt_s and Pkt_r represent the number of packets sent and received, and $Byte_s$ and $Byte_r$ represent the number of bytes sent and received, respectively. We then apply an agglomerative hierarchical clustering algorithm (described in detailed in the following paragraphs) to partition the set of vectors (i.e., of flows) $V_{tcp}(h) = \{v(h)_i\}_{i=1..|S_{tcp}(h)|}$ and $V_{udp}(h) = \{v(h)_i\}_{i=1..|S_{udp}(h)|}$ into a number of clusters. Each of the obtained clusters of flows, $C_j(h)$, represents a group of flows with similar size. For each $C_j(h)$ (notice that each vector can be mapped back to the flow it describes), we consider the set of destination IP addresses related to the flows in the clusters, and for each of these IPs we consider its `BGP` prefix (using `BGP` prefix announcements). Finally, we count the number of distinct `BGP` prefixes related to destination IPs in a cluster $bgp_j = BGP(C_j(h))$, and discard those clusters of flows for which $bgp_j < \Theta_{bgp}$.

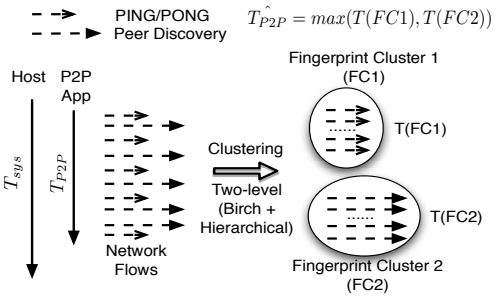


Figure 3: Example of Flow Clustering to Identify P2P Hosts

We call *fingerprint clusters* the remaining cluster of flows. Therefore, each host h can now be described by a set of fingerprint clusters $FC(h) = \{FC_1, .., FC_k\}$. We label h as P2P node if $FC(h) \neq \emptyset$, namely if h generated at least one fingerprint cluster.

The clustering algorithm for discovering clusters of similar flows may affect the system efficiency. For example, a direct usage of *hierarchical clustering* algorithm (with $O(n^2)$ time complexity where n is the number of flows) will introduce prohibitive time consumption when processing a large number of flows generated by a P2P node. Therefore, we design a two-level clustering scheme to improve its performance. First, given a pre-defined parameter Cnt_{birch}, we use BIRCH [23], a streaming clustering algorithm with time complexity $O(n)$, to efficiently identify at most Cnt_{birch} sub-clusters from the sets of TCP and UDP flows respectively, where the distance of two flows is defined as the Euclidean distance of $[Pkt_s, Pkt_r, Byte_s, Byte_r]$. Then, for each sub-cluster, we aggregate flows in it and represent it using a vector, where this vector describes the average value of each feature $[\overline{Pkt_s}, \overline{Pkt_r}, \overline{Byte_s}, \overline{Byte_r}]$ of flows in this sub-cluster. We further apply *hierarchical clustering* with *DaviesBouldin* validation index [7] on top of the vectors (sub-clusters), and find clusters of vectors, where each cluster represents a set of similar vectors (sub-clusters). For all the sub-clusters belonging to a cluster, we finally group the flows in these sub-clusters to the same cluster of flows. For this two-level clustering scheme, the time complexity to process the flows of one P2P node is mainly bounded by $O(Cnt_{birch}^2)$. Currently we configure $Cnt_{birch} = 4000$ (the evaluation of system performance over Cnt_{birch} is presented in Section IV-C5).

We applied this two-level clustering algorithm to the sample traces of 5 P2P clients. N_{bgp} in Table III presents the maximum number of distinct `BGP` prefixes of destination IPs in a fingerprint cluster. We therefore conservatively set the threshold $\Theta_{bgp} = 50$, which is much smaller than the measured N_{bgp}.

Figure 3 illustrates an example of the flow clustering process for a P2P node. Flows corresponding to `ping/pong` and `peer-discovery` share similar sizes respectively, and therefore they are grouped into two clusters (FC_1 and FC_2). Since the number of destination `BGP` prefixes

involved in each cluster is larger than Θ_{bgp}, we take FC_1 and FC_2 as its fingerprint clusters. A *fingerprint cluster summary*, ($\overline{Pkt_s}, \overline{Pkt_r}, \overline{Byte_s}, \overline{Byte_r}$, proto), presents the protocol and the average number of sent/received packets/bytes for all the flows in this fingerprint cluster. Examples of fingerprint cluster summaries for two `Bittorrent` traces (`T-Bittorrent` and `T-Bittorrent-2`) and one Skype trace, are illustrated in Table IV. "(1 1 145 319, UDP)" and "(1 1 109 100, UDP)" are shared by both `Bittorrent` sample traces. The payload of flows corresponding to these two fingerprint clusters are illustrated in Table V. It reveals that the fingerprint cluster of "(1 1 145 319, UDP)" represents the flows for node discovery, and that of "(1 1 109 100, UDP)" contains the flows for `ping/pong`.

Trace	Fingerprints
T-Bittorrent	1 1 145 319, UDP 1 1 109 100, UDP 1 1 146 340, UDP 5 3 346 170, TCP 1 1 145 310, UDP
T-Bittorrent-2	1 1 145.01 317.66, UDP 1 1 109 100, UDP 1 1 146 342, UDP 5 3 346 170, TCP 2 2 466 461, UDP

Trace	Fingerprints
Skype	1 1 74.58 60, UDP 1 1 78 60, UDP 1 1 75 60, UDP 1 1 76 60, UDP 1 1 79 60, UDP

Table IV: Summaries of Fingerprint Clusters

C. Identifying Persistent P2P Clients

As we mentioned at the beginning of Section III, P2P bots make themselves persistent into the compromised system, and run for as long as the system is powered on. Based on this observation, we aim to identify P2P clients that are active for a time T_{P2P} close to the active time T_{sys} of the underlying system they are running on. While this behavior *is not unique* of P2P bots and may be representative of other P2P applications (e.g., Skype clients that run for as long as a machine is on), identifying persistent P2P clients *takes us one step closer* to identifying P2P bots.

To estimate T_{sys} we proceed as follows. For each host $h \in \mathbf{H}$ that we identified as P2P clients according to Section III-B, we consider the timestamp $t_{start}(h)$ of the first network flow we observed from h and the timestamp $t_{end}(h)$ related to the last flow we have seen from h. Afterwards, we divide the time $t_{end}(h) - t_{start}(h)$ into w epochs (e.g., of one hour each), denoted as $T = [t_1, ..t_i, .., t_w]$. We further compute a vector $A(h, T) = [a_1, ..a_i, .., a_w]$ where a_i is equal to 1 if h generated any network traffic between t_{i-1} and t_i. We then estimate the active time of h as $T_{sys} = \sum_{i=1}^{w} a_i$.

The challenge is how to accurately estimate the active time of a P2P application. Since a P2P application periodically exchanges network control (e.g., ping/pong) messages with other peers as long as the P2P application is active, we can leverage the active time of a fingerprint cluster, which represents flows of control messages, in order to estimate the active time of the corresponding P2P application. For each host h (again, we consider only the hosts in $\mathbf{H}$, which we previously identified as P2P clients) we consider the set of its fingerprint clusters $FC(h) = \{FC_1, ..FC_j .., FC_k\}$ (see Section III), and for each fingerprint cluster FC_j we compute a vector $A(FC_j, T) = [a_1^j, ..a_i^j, .., a_w^j]$ where an element a_i^j is equal to 1 if the fingerprint cluster FC_j contains a flow between t_{i-1} and t_i, otherwise $a_i^j = 0$. We compute the active time of a fingerprint cluster FC_j as $T(FC_j) = \sum_{i=1}^{w} a_i^j$. Finally, we estimate the active time (T_{P2P}) of a P2P application as $\hat{T_{P2P}} = \max(T(FC_1), ..T(FC_j), ..T(FC_k))$.

If the ratio $r(h) = \frac{\hat{T_{P2P}}}{T_{sys}} > \Theta_{P2P}$, we say that h is running a persistent $P2P$ application, and add it to a set $\mathbf{P}$ of *candidate* P2P bots. Host h will then be input to our botnet detection algorithm (see Section III-D), where h will be represented by a set of persistent fingerprint clusters for h, denoted as $FC_p(h) = \{FC_i^1, .., FC_k^j\}$ where $T(FC_i) > \Theta_{P2P}$ for any $FC_i \in FC_p(h)$.

As illustrated in Table III, the estimated active time $\hat{T_{P2P}}$ is the same as the actual active time (T_{P2P}) of the P2P application, which demonstrates that $\hat{T_{P2P}}$ can accurately approximate T_{P2P}. As we can see from Table III, when we leave a P2P application running for as long as the machine is on (24 hours for this particular experiment) we obtain a ratio $r(h) = 1$. Therefore, we decided to conservatively set $\Theta_{P2P} = 0.5$. N_{clust} in Table III illustrates the size of $FC_p(h)$, the number of fingerprint clusters (FCs) whose $BGP(FC) > \Theta_{bgp}$ and $T(FC) > \Theta_{p2p}$.

D. P2P Botnet Detection Algorithm

Once we have identified the set $\mathbf{P}$ of candidate P2P bots, we apply our botnet detection algorithm. At this stage, our objective is to differentiate between legitimate persistent P2P clients and P2P bots. As we mentioned at the beginning of Section III, our detection approach is based on the following observations: i) bots that belong to the same botnet use the same P2P protocol and network, and ii) the set of peers contacted by two different bots have a large overlap, compared to peers contacted by two P2P clients connected to the same legitimate P2P network. Accordingly, we look for P2P clients that are running the same protocol and connect to the same P2P network, and whose sets of contacted destination IPs overlap significantly. We do so by introducing a measure of similarity between the fingerprint clusters, and then grouping P2P clients according to similarities between their respective *fingerprint clusters*.

We proceed as follows. For each host $h \in \mathbf{P}$, we consider the set of persistent fingerprint clusters $FC_p(h) = \{FC_1, .., FC_k\}$ (see Section III-B). For each $FC_i \in FC_p(h)$, we compute the average number of bytes sent, $\overline{Byte}_{s,i}$, and received, $\overline{Byte}_{r,i}$, in all flows in FC_i (remember that each fingerprint cluster FC_i is a cluster of flows). Also, for each cluster FC_i we extract the set of peers Π_i, i.e., the set of all destination IPs for the flows in FC_i. Therefore,

Fingerprints	flows	outgoing content	incoming content	description
1 1 145 319, UDP	1	d1:ad2:id20:. . . find_node1:. . . :y1:qe	d1:rd2:id20:. . . nodes208:. . . :y1:re	peer discovery
	2	d1:ad2:id20:. . . find_node1:. . . :y1:qe	d1:rd2:id20:. . . nodes208:. . . :y1:re	
	. . .	d1:ad2:id20:. . . find_node1:. . . :y1:qe	d1:rd2:id20:. . . nodes208:. . . :y1:re	
1 1 109 100, UDP	1	d1:ad2:id20:. . . :ping1:. . . :y1:qe	d1:rd2:id20:. . . :y1:re	ping/pong
	2	d1:ad2:id20:. . . :ping1:. . . :y1:qe	d1:rd2:id20:. . . :y1:re	
	. . .	d1:ad2:id20:. . . :ping1:. . . :y1:qe	d1:rd2:id20:. . . :y1:re	

Table V: Payload of flows in a fingerprint cluster of Bittorrent

each fingerprint cluster FC_i can be summarized by the tuple ($\overline{Byte}_{s,i}$, $\overline{Byte}_{r,i}$, Π_i). This allows us to define a notion of distance between fingerprint clusters. In practice, we define two separate distance functions as follows

i) $d_{bytes}(FC_i, FC_j) = \sqrt{(\overline{Byte}_{s,i} - \overline{Byte}_{s,j})^2 + (\overline{Byte}_{r,i} - \overline{Byte}_{r,j})^2}$

ii) $d_{IPs}(FC_i, FC_j) = 1 - \frac{|\Pi_i \cap \Pi_j|}{|\Pi_i \cup \Pi_j|}$

and then we define the distance between two hosts h_a and h_b as

$$dist(h_a, h_b) = \min_{i,j}\Big(\lambda * \frac{d_{bytes}(FC_i^{(a)}, FC_j^{(b)}) - min_B}{max_B - min_B} + (1-\lambda) * d_{IPs}(FC_i^{(a)}, FC_j^{(b)})\Big)$$

where

- $FC_k^{(x)}$ is the k-th fingerprint cluster of host h_x
- $min_B = \min_{i,j} d_{bytes}(FC_i^{(a)}, FC_j^{(b)})$
- $max_B = \max_{i,j} d_{bytes}(FC_i^{(a)}, FC_j^{(b)})$
- λ is a predefined constants, which we set to $\lambda = 0.5$.

After computing the distance between each pair of hosts (i.e., each pair of candidate P2P bots in set $\mathbf{P}$), we apply hierarchical clustering, and group together hosts according to the distance defined above. In practice, the hierarchical clustering algorithm will produce a dendrogram (a tree-like data structure) as shown in Figure 5. The dendrogram expresses the "relationship" between hosts. The closer two hosts are, the lower level they are connected at in the dendrogram. Two P2P bots in the same botnet should have small distance and thus are connected at lower level (forming a *dense* cluster). Even if these P2P bots' traffic is overlapped with traffic of legitimate P2P applications, the distance between two bot-compromised hosts is decided by the *minimum* distance of their respective *fingerprint clusters*. Since the distances of fingerprint clusters from botnet P2P protocols have smaller distance compared to those from legitimate P2P protocols (due to bots' large overlap of peer IPs), the minimum distance will stem from fingerprint clusters of P2P bots instead of legitimate P2P applications. Therefore, two bot-compromised hosts running legitimate P2P applications will still exhibit small distance. We then classify hosts in *dense* clusters as P2P bots, and discard all other clusters and the related hosts, which we classify as legitimate P2P clients. In practice, we cut the dendrogram at Θ_{bot} ($\Theta_{bot} \in [0, 1]$) of the maximum dendrogram height ($\Theta_{bot} * height_{max}$).

To set Θ_{bot}, we consider the following two assumptions: a) we assume we do not have a labeled data set of botnet traffic; b) we assume that the distance between two legitimate P2P applications is much larger than that between two bots belonging to the same botnet (as motivated above). Therefore, we conservatively set $\Theta_{bot} = 0.95$.

IV. Evaluation

In this section we present an evaluation of the effectiveness of our stealthy P2P botnet detection system.

A. Experimental Setup

We evaluated the performance of our detection system using real-world network traffic, including traffic collected from our academic network, traffic generated by popular P2P applications, and *live* P2P botnet traffic.

The traffic we collected from our academic network came from a span port mirroring all traffic crossing the gateway router (around 200-300Mbps) for the college networks. We used `Argus` [1] to efficiently collect network flow information of the traffic between internal and external networks for one entire day. Along with various flow statistics we also recorded the first 200 bytes of each *flow payload*, which we used to identify known legitimate P2P clients within our network. To reduce the volume and noise in our network traces, we excluded all traffic related to email servers, DNS servers, and `planetlab` nodes from our botnet detection analysis. The DNS traffic was collected simultaneously with the network flow information, using `dnscap`, to keep track of all the domain-to-IP mappings needed to perform traffic volume reduction. Overall, we observed 953 active hosts, as reported in Table VI. We refer to the traffic collected from our academic network as NET_{CoC}.

In order to establish some ground truth in terms of what hosts are running P2P applications, we used a signature-based approach by matching the signatures from [11] onto the first 200 bytes of each network flow. We further manually investigated each of these hosts to eliminate false positives (we found some spurious signature matches deriving from traffic towards SMTP servers that we were not able to pre-filter, and a few web requests towards our departmental website). After manual validation, we identified a total of 3 hosts that were running `Bittorrent` , which in the following we denoted as "BT1@C", "BT2@C" and "BT3@C". Furthermore, there exists no signature that can match P2P traffic generated by `Skype`, since `Skype` communications are encrypted. However, using the statistical traffic fingerprints, we were able to identify 5 likely `Skype` clients within our network (we discuss this in more detail in Section IV-C1), denoted as "Skype1@C", "Skype2@C", ..,

Trace	duration	# of TCP / UDP flows	# of clients
t-c	24h	61,745,989 / 20,226,837	953

Trace	duration	# of domains	# of IPs
t-dns	24h	328,965	268,753

Table VI: Traffic statistics for our academic network.

Trace	Dur	# of flows	# of Dst IPs	Avg Flow Size
Bittorrent-1/2	24 hr	250960/297785	17337/17657	68310/350205
Limewire-1/2	24 hr	229215/638103	11602/64994	1003/2038
Emule-1/2	24 hr	58941/110821	6649/14554	124267/22681
Skype-1/2	24 hr	88927/49541	10699/6264	514/1988
Ares-1/2	5 hr	17566/21756	1918/3118	69373/24755

Table VII: Traces of Popular P2P Applications

"Skype5@C". We refer to the network traces corresponding to these 8 P2P clients as $NET_{P2P@CoC}$. One possible reason why we found only a few (fewer than expected) P2P hosts is that our college network is well-managed and the usage of file sharing applications is highly discouraged. In addition, the vast majority of the hosts we have monitored are desktops managed by the college, where regular users have no permission to install software including `Skype`.

In order to increase the number and diversity of P2P nodes in our network, we ran 5 popular P2P applications, whose name and version are listed in Table I. We ran each of the 5 P2P applications in two different (virtual) hosts for several hours (e.g., 24 or 5 hours) *simultaneously*. Each host was represented by a WindowsXP (virtual) machine with a public IP address selected within a /24 network. Given a P2P application among the 5 we considered, we manually interacted with one instance (on one host) to simulate typical human-driven application usage behavior, and we fed the second instance of the application (on the second host) with automatically generated user-interface input. This artificial user input was simulated using a `AutoIt` [2] script that randomly selects contents to be downloaded or uploaded using the P2P application at random time intervals. Therefore, overall we obtained 10 additional network traces related to traffic generated by P2P applications (Table VII shows some statistics related to these network traces). We refer to these network traces as NET_{P2P}.

In addition, we were able to obtain network traces for two popular P2P botnets, `Storm` and `Waledac`. Both traces were collected by purposely running `Storm` and `Waledac` malware samples in a controlled environment, and recording their network behavior. The `Storm` traces included 13 different bot-compromised hosts, while the `Waledac` included 3 different bot-compromised hosts, as shown in Table VIII. It is worth noting that both traces were collected at a time when the two botnets were fully active, before any successful takedown attempt was carried out by law enforcement or network operators. We refer to these network traces as NET_{bots}.

B. Experimental Design

We structured our experiments in five parts:

1) Evaluate the effectiveness of identifying and profiling P2P applications using *statistical* fingerprint clusters. (see Section IV-C1)
2) Evaluate the detection performance by pretending that a number of machines in our network have been compromised with either `Storm` or `Waledac` (Section IV-C2).
3) Determine whether our system is able to detect P2P bots running on compromised machines that are also running legitimate P2P clients at the same time (Section IV-C3).
4) Estimate the detection performance in special cases, where only two bots or no bot (e.g., a "clean" network) appear in the monitored networks (Section IV-C4).
5) Analyze the effect of system parameters Cnt_{birch} and Θ_{bot} (Section IV-C5).

Trace	duration	size	# of bots
Waledac	24hr	1.1G	3
Storm	24hr	4.8G	13

Table VIII: Traces of Botnets

We prepared four data sets for evaluation, D_1, D_2, D'_1 and D'_2. We obtained D_1 as follows: For each host (denoted as h_{p2p}) of both 16 P2P bots (in NET_{bots}) and 10 P2P applications (in NET_{P2P}), we randomly selected one host (denoted as h_{CoC}) from trace NET_{CoC}, and we overlaid h_{p2p}'s traffic to the h_{CoC}'s traffic. We aligned the start time of the h_{CoC}'s traffic according to the start time of its corresponding h_{p2p}'s traffic. If the duration of h_{CoC}'s traffic was t_h and that of h_{p2p} was t_p, where $t_h > t_p$, we only kept the first t_p of h_{CoC}'s traffic. In effect, we simulated the scenario where the P2P bots/applications are running persistently in the underlying hosts. D_1 represents the scenario that a host is compromised by a P2P bot and some legitimate P2P applications are active in the same monitored network.

For D_2, we randomly selected half (8) of the P2P bots from NET_{bots}. Then for each of the 5 P2P applications we ran, we randomly selected one out of its two traces from NET_{P2P} and overlaid its traffic to the traffic of a randomly selected host from NET_{CoC}. We further randomly chose 3 P2P hosts from $NET_{P2P@CoC}$ identified in the first experiment (Section IV-C1). We finally overlaid each of 8 P2P bot traces to each of the selected 8 P2P traces (5 from NET_{P2P} and 3 from $NET_{P2P@CoC}$), as illustrated in the first two columns in Table IX. D_2 represents the scenario that a host, which is compromised by a P2P bot, has an active legitimate P2P application running at the same time.

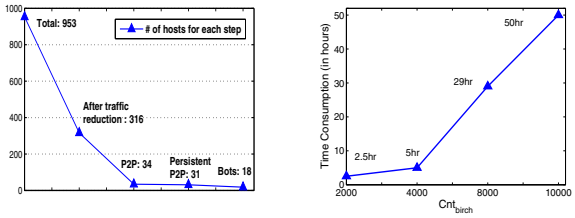
We use D'_1 to represent a "clean" network, where no host is compromised by P2P bots. We get D'_1 by simply removing all the hosts overlaid with bots' traces from NET_{bots}. In order to get D'_2, we randomly select hosts compromised by two bots for each botnet from D_2 and discard the rest of the hosts overlaid by the traces from NET_{bots}. So D'_2 represents the scenario in which only two bots from each botnet exist in the monitored network.

Bot	P2P App	Before Overlaying (Bot)			After Overlaying (Bot+P2PApp)		
		# of flows	# of DstIPs	avg flow size	# of flows	# of DstIPs	avg flow size
Waledac1	Emule1	341784	850	12829	452645	15338	55688
Waledac2	BT2@C	319119	760	11372	361135	1359	348708
Storm1	Limewire1	200237	6390	1342	429458	16635	1714
Storm2	BT3@C	275451	7319	1337	310667	8307	3381
Storm3	Bittorrent2	133955	5584	1344	432464	23261	172945
Storm4	Skype4@C	171471	7277	1280	199101	7520	1266
Storm5	Skype1	164917	6686	1328	214548	13137	1307
Storm6	Ares1	220459	6618	1307	238063	8543	6244

Table IX: Bot Traces Overlaid with P2P Application Traces

	TP	FP	Data	Description
1	100%	0.2%	D_1	bots overlaid with host
2	100%	0.2%	D_2	bots overlaid with P2P host
3	100%	0.2%	D_2'	only two bots
4	–	0.2%	D_1'	a "clean" network

Table X: Experimental Results



(a) Number of hosts identified by each step (on D_1)

(b) System Performance For Different Cnt_{birch}

Figure 4: Performance Evaluation

Trace	Fingerprints
BT1@C	1 1 109 100, UDP 1 1 109 91, UDP 1 1 104 178, UDP 1 1 319 145, UDP 1 1 145 319, UDP
BT2@C	1 1 145 319, UDP 1 1 75 75, UDP 1 1 65 65, UDP
BT3@C	7 6 1118 1767, TCP

Table XI: Fingerprint Cluster Summaries for 3 `Bittorrent` Clients

Trace	Fingerprints
Skype1@C	1 1 73 60, UDP 1 1 76 60, UDP 1 1 75 60, UDP 1 1 72 60, UDP 1 1 74 60, UDP
Skype2@C	1 1 75 60, UDP 1 1 74 60, UDP 1 1 76 60, UDP
Skype3@C	1 1 72 60, UDP 1 1 74 60, UDP 1 1 79 60, UDP 1 1 76 60, UDP
Skype4@C	1 1 73 60, UDP
Skype5@C	1 1 74 60, UDP 1 1 75 60, UDP

Table XII: Fingerprint Cluster Summaries for 5 Potential `Skype` Clients

Trace	Fingerprints	Trace	Fingerprints
Storm1	2 2 94 554, UDP 2 2 94 1014, UDP 2 2 94 278, UDP …	Storm2	2 2 94 554, UDP 2 2 94 1014, UDP 2 2 94 278, UDP …
Waledac1	4 3 224 170, TCP 3 3 186 162, TCP 5 4 286 224, TCP …	Waledac2	4 3 224 170, TCP 3 3 186 162, TCP 5 4 285 224, TCP …

Table XIII: Fingerprint Cluster Summaries for P2P Bots

C. Experimental Results

Table X summarizes the experimental results in Section IV-C2, IV-C3 and IV-C4, where we set the parameters as $\Theta_{bot} = 0.95$ and $Cnt_{birch} = 4000$. The effect of varying Θ_{bot} and Cnt_{birch} is discussed in Section IV-C5.

1) Identifying and Profiling P2P Applications

We applied our detection system on data set D_1. The number of hosts kept after each step is presented in Figure 4(a). Traffic reduction using DNS traffic significantly reduced the number of hosts and the number of flows we needed to process, thereby greatly reducing the workload for the coming steps. For example, as illustrated in Figure 4(a), other components only need to process approximately one-third of the hosts (316 out of 953) after traffic reduction.

Our system identified 34 hosts as P2P clients in total. These 34 hosts are composed of i) *all* 16 P2P bots, ii) *all* 10 hosts with 5 popular P2P applications we have tested, and iii) 8 other hosts in the college networks. For those 8 hosts, 3 are `Bittorrent`-related hosts (a.k.a, BT1@C, BT2@C and BT3@C), which have been verified by the content-based signatures. The remaining 5 identified hosts do not match any content-based signature. We present their fingerprint cluster summaries ($\overline{Pkt_s}, \overline{Pkt_r}, \overline{Byte_s}, \overline{Byte_r}$, proto) in Table XI and Table XII.

The fingerprint cluster summaries for 3 `Bittorrent` clients are presented in Table XI. For BT1@C and BT2@C, "(1 1 145 319 UDP)" is consistent with one fingerprint cluster of a sample `Bittorrent` trace described in Table IV. The fingerprint of BT3@C is different from other two, which may represent another version implementation of the `Bittorrent` protocol.

The fingerprint cluster summaries for the remaining 5 unknown P2P hosts are presented in Table XII. By referring to Table IV, their fingerprint cluster summaries are very close to those of the `Skype` trace. For example, "(1 1 75 60, UDP)" is shared by most of these clients and the sample `Skype` traffic. This indicates that these 5 hosts are mostly likely `Skype` clients.

Some fingerprint cluster summaries for `Storm` and `Waledac` are presented in Table XIII. P2P bots in the same botnet exhibit great similarity on fingerprint clusters, while their fingerprint clusters are different compared to those of another P2P botnet and legitimate P2P applications (e.g., `Bittorrent` and `Skype`). We apply our system on D_2 to investigate whether our system can effectively profile different P2P applications if a bot-compromised host is also running a legitimate P2P application. Table XIV presents several fingerprint cluster summaries for two bots overlaid with legitimate P2P applications, `Waledac2+BT2@C` and `Storm4+Skype4@C`. For the example of `Waledac2+BT2@C`, we can find that its fingerprint clusters come from two applications, where "(1 1 145

Trace	Fingerprints
Waledac2+BT2@C	1 1 145 319, UDP (Bittorrent) 4 3 224 170, TCP (Waledac) 3 3 185 162, TCP (Waledac) 1 1 75 75, UDP (Bittorrent) ...
Storm4+Skype4@C	2 2 94 554, UDP (Storm) 2 2 94 1014, UDP (Storm) 1 1 73 60, UDP (Skype) ...

Table XIV: Fingerprints for `Storm` and `Waledac`

		Θ_{bot}						
Cnt_{birch}	-	0.1	0.3	0.5	0.7	0.8	0.9	0.95
2000	DR	0	0	2/16	3/16	16/16	16/16	16/16
	FP	0	0	0	0	0	0	2/953
4000	DR	2/16	3/16	3/16	16/16	16/16	16/16	16/16
	FP	0	0	0	0	0	0	2/953
8000	DR	2/16	3/16	3/16	16/16	16/16	16/16	16/16
	FP	0	0	0	0	0	0	2/953
10000	DR	2/16	3/16	3/16	16/16	16/16	16/16	16/16
	FP	0	0	0	0	0	0	2/953

Table XV: Detection Rate and False Positive Rate for Different Θ_{bot} and Cnt_{birch}

139, UDP)" and "(1 1 75 75, UDP)" are from `Bittorrent` protocol (referring to the second row in Table XI), and "(4 3 224 170, TCP)" together with "(3 3 185 162, TCP)" are from `Waledac` (referring to Table XIII).

These experimental results demonstrate that our system can effectively identify hosts engaging in P2P communications. In addition, the generated fingerprint clusters can effectively profile P2P applications.

2) Detecting P2P Bots

We applied our system on D_1 to detect P2P bots. As we discuss in Section IV-C1, the system identified 34 P2P hosts. By estimating the active time of the P2P application for each of the 34 hosts, our system identified 31 hosts exhibiting *persistent* P2P communications.

For these 31 hosts, our system constructs a hierarchical tree (Figure 5(a)) by evaluating the distance ($dist(h_a, h_b)$ defined in Section III-D) between P2P hosts. P2P bots share same P2P protocol and have large overlap of the peer IP addresses in fingerprint clusters, thereby resulting in small distances and dense clusters in consequence. As shown in the Figure 5(a), both `Storm` and `Waledac` bots have small distances to each other and form dense clusters respectively. We cut the tree at $\Theta_{bot} * height_{max} = 0.475$ ($\Theta_{bot} = 0.95$) to identify dense clusters. As a consequence, three clusters are identified and therefore a total of 18 hosts were labeled as suspicious. All 16 P2P bots were detected, resulting in a high detection rate of 100% and a low false positive rate of 0.2% ($2/953$). The false positives appear to be two `Skype` clients. The reason for these two false positives is the conservatively configured value of Θ_{bot}, which is close to 1.

3) Detecting P2P Bots Overlaid with P2P Applications

We applied our detection system on data set D_2 to evaluate the detection accuracy when a bot-compromised host happens to run a legitimate P2P application. Table IX presents some statistics of the bot traces *before* and *after* overlaying legitimate P2P application traces. Some of bot-compromised hosts' traffic profiles are significantly distorted after traffic overlaying. For example, after overlaying `BT2@C` (a real P2P client identified in the college network) traffic to the `Waledac2` traffic, the average flow size is increased from 11372 to 348708 and the number of destination IP addresses, which are involved in the successful outgoing connections, is also increased from 760 to 1359. It is because the `Bittorrent` application could be actively used for downloading/uploading files, thereby dominating the traffic profile of the host. In this case, if we use the traffic profile of the *entire* host (e.g., the average flow size and number of destination IP addresses) to detect the bot, the bot behavior will be concealed by the `Bittorrent` traffic. As a consequence, the detection approaches such as [19], which use the traffic profile of the *entire* host for detection, will lose effectiveness.

However, since our system leverages fine-grained information of *fingerprint clusters*, which describe the profiles of *P2P applications* instead of *entire host*, it can still detect bots even if their underlying hosts are running legitimate P2P applications. The hierarchical tree for detection is presented in Figure 5(b), where `Waledac` bots and `Storm` bots still form dense clusters. Compared to the hierarchical tree in Figure 5(a), the tree structure in Figure 5(b) stays stable, which is not affected by the overlaid legitimate P2P applications. It is because the distance of two bot-compromised hosts is based on the minimum distance of fingerprint clusters from two P2P bots, the new fingerprint clusters introduced by the P2P application would not affect the minimum distance.

In D_2, our system identified 26 P2P clients, where 25 out of them exhibit persistent P2P behaviors. With $\Theta_{bot} = 0.95$, we cut the tree at 0.475, and identify three groups of hosts (18 in total). Among these 18 suspicious hosts, all 16 P2P bots are successfully identified with a low false positive rate (0.2%). The detection result is not affected by the overlaid traffic from legitimate P2P applications. This demonstrates that our system can effectively detect bots even if bot-compromised hosts run legitimate P2P applications.

4) Detection Performance in Special Cases

It is possible that in the monitored network, only two hosts are compromised by bots from the same botnet. We applied our system on data set D_2', and achieved the detection rate of 100% and false positive rate of 0.2%.

It is also possible that the monitored network is "clean", where no host is compromised by P2P bot. In this case, the false positive is a concern. We applied our system on D_1', which simulates a "clean" network environment, where we get a low false positive rate of 0.2%.

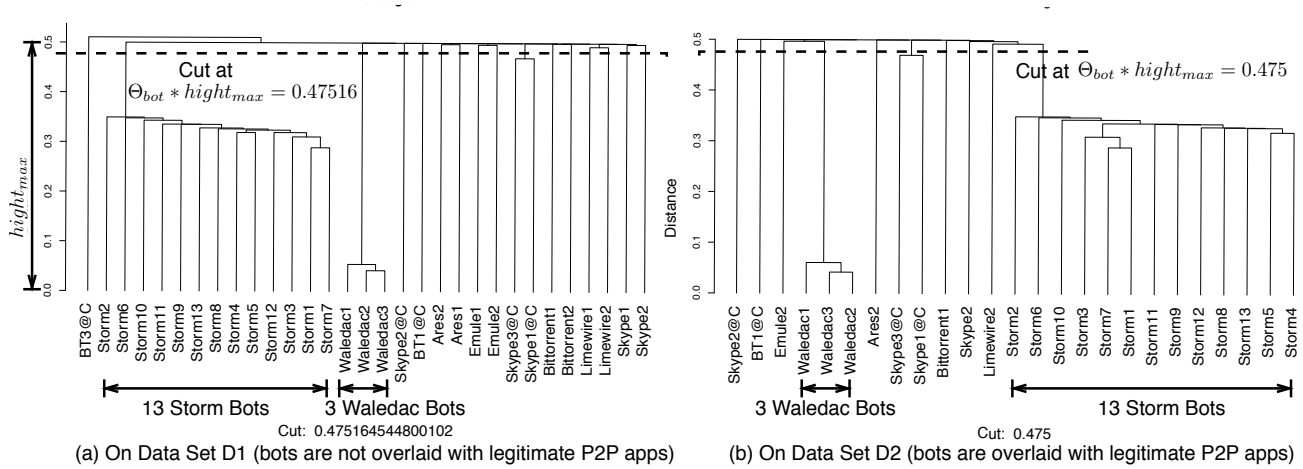


Figure 5: Hierarchical Tree on Persistent P2P Hosts

5) Analyzing The Effect of System Parameters

While the measurement in Section III motivates the parameter values for Θ_o, Θ_{bpg} and Θ_{p2p}, we study system parameters Cnt_{birch} and Θ_{bot} in this section. Cnt_{birch} may introduce a trade-off between system efficiency and effectiveness. For example, by decreasing Cnt_{birch}, the system has less vectors to process in *Hierarchical clustering* and thus increase the system efficiency. However, a small Cnt_{birch} may force dissimilar flows to be aggregated into the same sub-cluster and therefore into the same fingerprint cluster, resulting in inaccurate fingerprint clusters.

To evaluate Cnt_{birch} and Θ_{bot}, we conducted the following experiments. We applied our system D_2 with different Cnt_{birch} values, including 2000, 4000, 8000 and 10000. The time consumption of our system is presented in Figure 4(b), which demonstrates a significant efficiency improvement as Cnt_{birch} decreases. For each Cnt_{birch} value, we further adopted different Θ_{bot} (i.e., 0.1, 0.3..0.95) values to evaluate the detection rate and false positive rate. The results of detection rate (DR) and false positive (FP) rate are described in Table XV. The experimental results indicate that: 1) The two-level clustering scheme can greatly increase the system efficiency. For example, $Cnt_{birch} = 4000$ enables a reduction of time consumption by 90% compared to $Cnt_{birch} = 10000$ without sacrificing the detection accuracy. 2) The detection performance is stable over a large range of Cnt_{birch} (e.g., ≥ 4000) and $\Theta_{bot} \in [0.7, 0.95]$ is a good candidate value. This experiment also suggests that 0.8 or 0.9 may be a better value for Θ_{bot}. This implies that when a labeled data set of P2P botnet traffic is available we can tune this threshold (Θ_{bot}) to find a better trade-off between false positives and false negatives.

In summary, our system can effectively detect all the P2P bots with a very low false positive rate, even if the bot-compromised hosts are running legitimate P2P applications. Our system is stable over a large range of values for system parameters and shows great efficiency.

V. Discussion

For practical deployment, the system can be configured to automatically run daily. In this case, `Argus` and `dnscap` collect flow and DNS data in real-time and our detection system analyzes the data in batches at the end of each day. The memory consumption is mainly constrained by the maximum number of flows per host. And the time consumption is mainly bounded by $N_{host} * O(Cnt^2_{birch})$ (for the flow-clustering-based analysis), where N_{host} is the number of hosts in the monitored network.

If botmasters get to know about our detection algorithm, they could attempt to modify their bots' network behavior to evade detection. This situation is analogous to evasion attacks against other intrusion detection systems. Since our detection algorithm is based on differentiating P2P protocols used by P2P bots from legitimate P2P applications, botmasters may instruct the bots to join existing legitimate P2P networks, and use legitimate P2P networks to propagate commands. The initial version of `Storm` adopted this strategy. However, such approach exposes the botnet to *sybil* attacks, where researchers can infiltrate the P2P network and enumerate/detect the bots [8]. Therefore, current P2P botnet, including `Storm` and `Waledac`, isolate their own P2P network from existing legitimate P2P networks. Botmasters may leverage our traffic volume reduction component to evade detection. For example, the botmaster may set up a malicious DNS server, and instruct each bot to query this server before contacting any peer, asking the malicious DNS server to return a response containing the peer's IP address. In this case, our traffic reduction component would eliminate the corresponding flows from the analysis. To avoid this evasion attempt, we could filter traffic based only on DNS responses for popular domains, i.e., domains queried by a non-negligible fraction of hosts in the monitored networks. Bots could also intentionally try to reduce the number of

contacted peer IPs (or BGP prefixes) or the active time of the bot, in order to bypass the P2P client identification or the component that detects persistent P2P applications. However, such techniques could have a serious negative impact on the resiliency of the C&C infrastructure and limit the usability of the entire botnet. Another evasion approach could exploit the Θ_{p2p} threshold. For example, the P2P bots could exchange traffic for a short period of time, then go idle for several hours, and repeat this pattern. However, this evasion technique is equivalent to increasing the churn rate for the P2P nodes, which may eventually bring to a complete disruption of the overlay network [4]. Bots could also randomize their P2P communication patterns to prevent our system from getting an accurate profile of P2P protocols. For example, bots could inject noise into network flows related to P2P control messages. In this case, we could use other features (e.g., the distribution of flow sizes) to profile the P2P protocols. A P2P botnet could also attempt to reduce the overlap between peers contacted by the bots. For example, the botnet could partition the peers into different sets and ask each bot to contact disjoint sets of peers. Such technique may require a lot of efforts for the design and operation of the P2P botnets. We leave the analysis of such complex botnets to future work. We should always strive to develop more robust defense techniques. Combining different complementary detection techniques to make the evasion harder is one of the possible directions that we intend to explore in our future work.

VI. Conclusion

In this paper, we presented a novel botnet detection system that is able to identify *stealthy* P2P botnets. Our system aims to detect all P2P botnets, even in the case in which their malicious activities may not be observable. To accomplish this task, we first identify all hosts within a monitored network that appear to be engaging in P2P communications. Then, we derive *statistical fingerprint*s of the P2P communications generated by these hosts, and leverage the obtained fingerprints to distinguish between hosts that are part of legitimate P2P networks (e.g., file-sharing networks) and P2P bots. We implemented a prototype version of our system, and performed an extensive experimental evaluation. Our experimental results confirm that the proposed system can detect stealthy P2P bots with a high detection rate and a low false positive rate.

Acknowledgments

We thank Paul Royal for the help in collecting network traces. This material is based upon work supported in part by the National Science Foundation under grant no. 0831300, the Department of Homeland Security under contract no. FA8750-08-2-0141, the Office of Naval Research under grants no. N000140710907 and no. N000140911042. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation, the Department of Homeland Security, or the Office of Naval Research.

References

[1] Argus: Auditing network activity. http://www.qosient.com/argus/.

[2] Autoit script. http://www.autoitscript.com/autoit3/index.shtml.

[3] A. W. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS*, 2005.

[4] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *ACM SIGCOMM Internet Measurement Conf*, 2006.

[5] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through IDS-driven dialog correlation. In *Proc. USENIX Security*, 2007.

[6] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proc. USENIX Security*, 2008.

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

[8] 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 *Proc. USENIX LEET*, 2008.

[9] B. Kang, E. C. Tin, and C. P. Lee. Towards complete node enumeration in a peer-to-peer botnet. In *Proc. ACM ASIACCS*, 2009.

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

[11] Z. Li, A. Goyal, Y. Chen, and A. Kuzmanovic. Measurement and diagnosis of address misconfigured p2p traffic. In *IEEE INFOCOM 2010*, 2010.

[12] M.P. Collins and M. K. Reiter. Finding peer-to-peer file sharing using coarse network behaviors. In *Proc. ESORICS*, 2006.

[13] P. Porras, H. Saidi, and V. Yegneswaran. A multi-perspective analysis of the storm (peacomm) worm. In *Computer Science Laboratory, SRI International, Technical Report*, 2007.

[14] P. Porras, H. Saidi, and V. Yegneswaran. Conficker c analysis. http://mtc.sri.com/Conficker/addendumC/index.html, 2009.

[15] S. Nagaraja and P. Mittal and C.-Y. Hong and M. Caesar and N. Borisov. Botgrep: Finding p2p bots with structured graph analysis. In *Proc. USENIX Security*, 2010.

[16] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identication of p2p traffic using application signatures. In *WWW*, 2004.

[17] G. Sinclair, C. Nunnery, and B. B. Kang. The waledac protocol: The how and why. In *Intl. Conf. Malicious and Unwanted Software*, 2009.

[18] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the storm and nugache trojans: P2p is here. In *USENIX; login, vol. 32, no. 6*, 2007.

[19] T.-F. Yen and M. K. Reiter. Are your hosts trading or plotting? telling p2p file-sharing and bots apart. In *ICDCS*, 2010.

[20] T. Karagiannis, A.Broido, M. Faloutsos, and Kc Claffy. Transport layer identification of p2p traffic. In *ACM IMC*, 2004.

[21] T. Karagiannis and K. Papagiannaki and M. Faloutsos . Blinc: Multilevel traffic classification in the dark. In *ACM SIGCOMM*, 2005.

[22] Y. Zhao and Y. Xie and F. Yu and Q. Ke and Y. Yu. Botgraph: Large scale spamming botnet detection. In *Proc. USENIX NSDI*, 2009.

[23] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proc. ACM SIGMOD*. ACM Press, 1996.