BotSuer: Suing Stealthy P2P Bots in Network Traffic through Netflow Analysis

Nizar Kheir[1] and Chirine Wolley[2]

[1] Orange Labs, Issy-Les-Moulineaux, France
nizar.kheir@orange.com
[2] Aix-Marseille University, LSIS UMR 7296, Marseille, France
chirine.wolley@etu.univ-amu.fr

Abstract. A large proportion of modern botnets are currently shifting towards structured overlay topologies, using P2P protocols, for command and control. These topologies provide a better resilience against detection and takedown as they avoid single nodes of failure in the botnet architecture. Yet current state of the art techniques to detect P2P bots mostly rely on swarm effects. They detect bots only when there is multiple infected nodes belonging to the same botnet inside a network perimeter. Indeed, they cannot detect botnets that use public P2P networks such as the TDSS malware using Kad, let alone botnets that encapsulate P2P overlays within HTTP traffic, such as waledac, or even hide behind Tor networks.

In this paper, we propose a new and fully behavioral approach to detect P2P bots inside a network perimeter. Our approach observes only high-level malware traffic features with no need of deep packet inspection. We run samples of P2P malware inside a sandbox and we collect statistical features about malware traffic. We further use machine learning techniques in order to first clean the features set by discarding benign-like malware P2P behavior, and second to build an appropriate detection model. Our experimental results prove that we are able to accurately detect single infected P2P bots, while also satisfying a very low false positives rate.

Keywords: Botnet detection, P2P malware, machine learning, netflow.

1 Introduction

Malware has recently become the mainstream arsenal for cyber attackers, including thousands of malware samples being created every day [25]. It embeds zero-day exploits, self replication mechanisms, propagation capabilities, and more importantly it connects to a master server as to retrieve commands or to send stolen data. Arguably, botnets are the most common type of malware today. These are networks of infected nodes (bots) that are controlled by the same entity (botmaster) via shared Command and Control (C&C) channels. Early botnet C&C channels were mostly centralized, using protocols such as `IRC` and `HTTP` [19]. Although being easy to manage and highly responsive, centralized

C&C channels are susceptible to detection and takedown because they include single nodes of failure in their botnet architecture. Therefore, botnet C&C channels started shifting towards decentralized architectures that use peer-to-peer protocols (P2P) [21,23]. P2P botnets are robust because they constitute overlay networks where bots can distribute commands without the need for a central C&C server [10].

Nowadays, botnets are no longer being used only to trigger massive distributed attacks such as spam and DDoS. They are becoming stealthier in the way they perform malicious activities, seeking financial benefits and sensitive user data [27]. Botnet herders also use code obfuscation in order to hide malware payloads within the large amount of network traffic [18], and so to evade IDS signatures. Network detection solutions are thus increasingly confronted to *stealthy bots*, along with *encrypted C&C communications* and *only few infections* inside a network perimeter. Yet the use of overlay networks makes detection more difficult due to the inability to build blacklists of URLs or malicious domain names, as for centralized botnet topologies [5]. We thus derive four main requirements for network-based systems in order to detect P2P bots. First, they should be able to detect bots even when no malicious activity is being observed. Besides, they should be able to detect even single infected bots inside a network perimeter. They should also detect P2P bots using only network layer features, without the need to access encrypted packet payloads. Last of all, they should detect P2P bots based on their overall network footprints, and not based on every single connection. While the first three requirements look straightforward, the fourth one stems from the fact that P2P bots locate and retrieve commands using the overlay network. Two infected bots are thus unlikely to connect to the same set of peers, even though some overlaps may occur. Therefore, the system should be able to detect P2P infected nodes based on the way these nodes interact with the overlay network, and not based on the single IP addresses being contacted.

This paper presents BotSuer, a system that detects P2P bots by monitoring traffic inside a network perimeter. BotSuer leverages the fact that malware belonging to the same family communicates with the overlay P2P botnet in a similar way. In fact, P2P control flows implement multiple functionalities such as keep-alive, route discovery and data queries. As shown in [14], flow size distributions exhibit discontinuities almost for all P2P protocols. Such discontinuities characterize clusters of flows that implement the same P2P functionality, and so they would have similar network behavior in terms of flow size, number of packets and flow duration. While certain clusters may be common for both malware and benign P2P communications (e.g. keep-alive messages), others clearly show differences that can be accounted for during detection. For instance, data search queries for the Zeus P2P botnet show periodicities that are unlikely to appear in other benign kademlia P2P traffic [12]. Yet we observed a significantly lower chunk rate for route discovery requests triggered by a Sality botnet, compared to other benign P2P applications.

BotSuer monitors network traffic triggered by P2P malware samples running in a controlled environment. It builds a training set of malware P2P traffic by first filtering non-P2P traffic, and then clustering malware P2P flows in order to group together those that implement the same P2P functionality (e.g. keep-alive, route discovery, data search and push requests). Some malware P2P clusters are likely to appear in benign P2P traffic and these should be discarded as they cannot be used during detection. For instance, keep-alive flows triggered by a TDSS malware that uses the kad network are similar to those triggered by benign P2P applications that implement the same kademlia protocol. BotSuer thus correlates cluster footprints for both malware and benign P2P applications in order to discard non-discriminatory P2P clusters. It uses remaining clusters as a training set in order to build a P2P botnet detection model.

We evaluated BotSuer against real-world P2P traffic, including both botnet and benign P2P applications. We obtained samples of P2P malware, all being active in the wild by the time they were collected, and that implement different P2P protocols. We also tested our system against traffic collected from a corporate network, as well as anonymized traffic collected from a large ISP network. Our experimental results prove the ability of BotSuer to accurately differentiate malware and benign P2P applications, with only few false positives.

This paper will be structured as follows. Section 2 summarizes related work. Section 3 provides an overview of our system. Section 4 describes the workflow and the different modules that constitute BotSuer. Section 5 presents our experiments and main results. Section 6 discusses the limitations and provides future work. Finally, section 7 concludes.

2 Related Work

Related work includes several approaches that detect P2P botnets by monitoring network traffic. First of all, solutions such as BotGrep [16], BotTrack [9] and BotMiner [11] correlate netflow data [6] and localize P2P bots based on their overlay C&C topologies. They cluster hosts in order to isolate groups of hosts that form P2P networks. Then they separate malicious and benign P2P groups using information about infected nodes collected from multiple sources such as honeypots and intrusion detection systems. However, botnet activity is becoming stealthier and cannot be easily detected by IDS signatures, thus limiting the coverage of these techniques.

Another trend of research aims at detecting infected P2P bots inside a given network perimeter [29,30]. In this category, Yen et al. [29] discard benign P2P applications based on features such as the volume of P2P traffic and the persistence of P2P applications. Unfortunately, these features do not reliably separate benign and botnet P2P flows. In fact, authors rule out the possibility that certain benign P2P applications such as skype may not implement P2P file sharing and so they would be difficult to separate from malicious P2P flows. On the other hand, Zhang et al. [30] propose an alternative approach using only P2P control flows. This approach is similar to ours by means of using only control flows for

P2P botnet detection. However, it detects P2P bots only when there is multiple infected nodes belonging to the same botnet. It computes distances between clusters of flows triggered by different nodes in the network. It also uses flow sizes and contacted IP addresses as a basis to compute distances between P2P nodes. Indeed we believe these metrics would generate false positives in case of popular P2P applications such as skype. For example, we observed strong overlap between the remote IP addresses contacted by two skype clients running on two different nodes in the same network. Hence, the approach presented in this paper provides a better alternative as it only relies on the way bots interact with their overlay C&C networks, and not on the single remote IP addresses being contacted.

Last of all, Bilge et al. propose an alternative approach to detect botnets through large scale netflow analysis [4]. This approach is similar to ours as it processes and correlates netflow records in order to detect infected bots. Yet it observes traffic at large ISP networks and detects only central C&C servers. Therefore, it is efficient only against centralized botnet architectures, but does not detect distributed P2P botnets.

3 System Overview

Our system operates in two phases, the training phase and the detection phase, as illustrated in Fig. 1. The training phase builds a behavioral model using a dataset of malware and benign P2P traffic. The detection phase applies the behavioral model on network traffic in order to detect P2P bots.

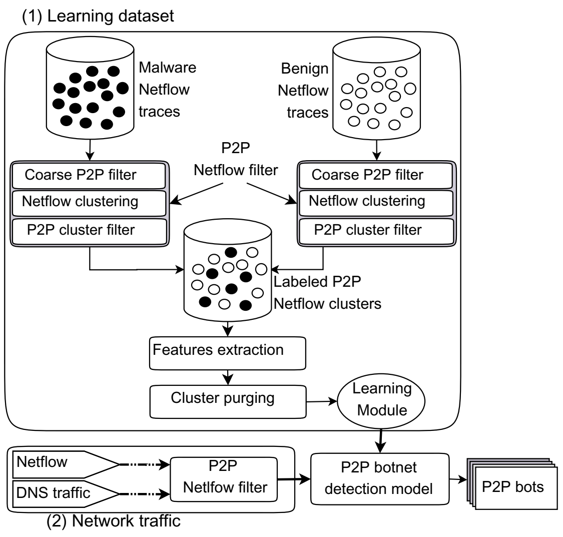

Fig. 1. Botnet Detection System

In the training phase, malicious traffic is obtained by executing malware in a controlled environment such as Anubis[2], CWSandbox[28] or Cuckoobox[1]. We pick-up P2P malware samples by checking antivirus (AV) signatures using the virusTotal API[1]. Such malware often triggers non-P2P traffic in addition to its C&C communications. We thus propose a filtering module that discards non-P2P traffic and keeps only P2P flows as input to the training phase. Our P2P filter implements two levels of filtering, including coarse and fine-grained filtering. Coarse filtering discards flows using high-level features such as DNS traffic and chunk rates. For example, overlay networks usually operate outside the DNS system, so we discard flows preceded with a successful DNS resolution as not being P2P flows. After the coarse filter, we cluster remaining flows for each malware sample in order to group together flows that are similar enough to be considered as part of the same application (e.g. spam campaign, scan, P2P keep alive, P2P search queries). We use statistical features such as flow size, number of packets, and the average bit rate. Control flows that implement the same P2P functionality for a given P2P protocol have similar network features and so they will be grouped within same clusters. The fine filter further eliminates clusters that do not implement P2P functionalities, and keeps only P2P clusters as input to train our detection module.

Benign P2P flows are more difficult to obtain as we lack the ground truth about the legitimate behavior of these applications. In this paper, we collected netflow packets from a well-protected corporate network. Terminals connected to this network abide to strict security policies, and they are all equiped with updated AV softwares. Access to this network is monitored using a proxy server with SSL inspection capability. Therefore, it is fairly reasonable to consider as benign all traffic collected from this network. Of course we cannot rule-out the possibility of few terminals being infected. However, these would be limited if compared to the overall amount of traffic and so they would have little impact on our detection model. Then we discard non-P2P flows and we build benign P2P clusters, using our P2P filter, the same way as we did for malware traffic.

The output of this process is a labeled set of malware and benign P2P flow clusters. We further propose a comprehensive set of features that we use in order to compute a network footprint for each cluster, including time, space and flow size-based features. Time features capture unusual sequences of flows and periodicities in a given cluster. Space features capture the dynamics of a P2P network, including geographical distribution and chunk rate, which is the rate of (new) IP addresses and ports contacted by a P2P application. Last of all, size-based features capture high-level flow metrics such as the amount of bytes and packets being shared by flows in a given cluster. We use cluster footprints in order to first purge the training set by eliminating non-discriminatory P2P clusters, and then to build our detection model. We propose a cross-correlation process that discards P2P clusters that have similar footprints in both malware and benign P2P applications. These are mostly keep-alive flow clusters that are likely to have similar implementations in both categories of P2P application. We

[1] `http://www.virustotal.com/`

further use remaining cluster footprints as input to train and build our detection model. In the detection phase, BotSuer applies this model on network traffic in order to detect P2P bots, without the need to inspect packet payloads. In the following section we describe our system and we build our detection model.

4 System Description

4.1 P2P Training Set

BotSuer processes P2P flows obtained during malware execution in a sandbox. During the buildup phase, we discard non P2P malware by checking AV signatures in virusTotal. Sure we may keep non P2P malware in our dataset due to misclassifications in virusTotal, but this malware will be dropped by our system as seen further in this section.

Coarse P2P filter applies to both malware and benign flows. It discards as many non-P2P flows prior to flow clustering and the fine P2P filter.

First, it discards flows preceded with a successful DNS resolution. P2P networks constitute environments of unpredictable IP addresses where nodes constantly join and leave the network. Peering nodes contact other peers using routing tables for the overlay network, with no prior DNS resolution. When access to a central server through DNS resolution is still possible at bootstrap, nodes further communicate using IP addresses in the overlay network [3].

Second, the churn effect is an inherent property of P2P systems and critical to their design and implementation [24]. It is a direct consequence of the independent arrival and departure by thousands of peers in the network, and results in a significant rate of failed connection attempts. We use this rate within malware and benign traffic in order to discard those that do not implement P2P applications. We set a threshold τ_{fc} for the ratio of failed flows with respect to the total number of flows triggered by a malware or a network terminal. We empirically set τ_{fc} to 0.15, based on P2P malware samples that we observed in our dataset. Our filter drops all malware and benign terminals whose rate of failed connection attempts does not exceed this threshold.

Netflow Clustering: The coarse filter significantly reduces the volume of input data by discarding flows that are unlikely to implement P2P applications. However, other non-P2P flows that have similar properties, such as spam and scan, may also match this filter. Indeed, we want to *cluster flows for a given malware or benign terminal* in order to put together flows that are likely to implement the same functionality, and then to discard non-P2P clusters. As in [14], control flows for a given P2P protocol usually implement preferred packet sizes, resulting in similar flows when observed at the network layer. These flows are grouped into categories where they implement the same P2P control activity. We propose a clustering process where flows that implement similar functionalities for a given malware or benign terminal are grouped within the same clusters. The fine filter further keeps only P2P clusters as input to our learning module. In fact we cluster flows using high level features, without access to packet payloads. We represent a flow f using the following features vector V_f:

$$V_f =< Pckt_s, Pckt_r, Byte_s, Byte_r, BRate_s, BRate_r >$$

$Pckt_s$ and $Pckt_r$ respectively represent the number of packets sent and received; $Byte_s$ and $Byte_r$ respectively represent the number of bytes sent and received; and $BRate_s$ and $BRate_r$ represent the byte rate sent and received. We define the distance between two flows as the Euclidian distance between their respective feature vectors. Then we apply the unsupervised, incremental k-means [20] to cluster flows $\mathcal{F}_\alpha$ triggered by malware M_α or a terminal T_α. Incremental k-means is a fast algorithm that requires no prior knowledge about the number of clusters, which is a key requirement to our approach. It provides a better alternative to the hierarchical clustering as we can set the threshold to create a new cluster based on our learning set of malware and benign P2P flows. In fact, incremental k-means creates a new cluster when the distance of an entry to all existing clusters exceeds a given threshold (τ_{cl}). We tested the incremental k-means algorithm against our set of malware P2P flows in order to find the optimal threshold τ_{cl}. We manually checked the outcome of the clustering process for different values of τ_{cl}. We thus empirically set τ_{cl} to the value 60, that we found to give the best output clusters (section 5.1 discusses more in details the impact of this threshold).

Fine P2P Filter implements two levels of filtering, using the distributions of autonomous systems and destination ports contacted by flows in each cluster.

Filtering by destination ports distribution: P2P applications usually hide while using nonstandard ports [13]. The use of multiple and oftentimes random destination ports is a distinctive P2P characteristic, as opposed to other activities such as spam or scan. For instance, a 10 minutes netflow trace includes at least 180 distinct destination port in case of skype, and almost 4690 distinct ports in case of bittorrent. We discard non-P2P clusters based on the distribution of new ports contacted. We compute the duration of each cluster, which is the lapse of time between the first and the last flow in the cluster. We split this interval into n sub-intervals of equal lengths. We compute, for each sub-interval, the number of new destination ports, that is the destination ports not appearing in previous sub-intervals. Then we compute the mean value for this distribution within each cluster, and we discard clusters where this value does not exceed a threshold τ_{np}. We conservatively set the value of τ_{np} using our dataset of malware P2P traffic, and that corresponds to the value $\tau_{np} = 0.6$.

Filtering by destination AS: Certain P2P malware, such as waledac, may bypass our port-based filter as it encapsulates its P2P activity within HTTP traffic, using the tcp port 80 [26]. However, the overlying P2P network still constitutes an overlay architecture that connects nodes distributed on multiple autonomous systems (AS). We thus use the number of contacted AS as another distinctive feature of P2P control clusters. Note that other clusters, mostly those containing malware scan flows, may also include a large number of destination AS. However, these clusters include a large number of flows, resulting in a low ratio of destination AS with respect to the number of flows in a cluster. We define the ratio of contacted AS in a cluster c as $\mathcal{R}_{AS_c} = \frac{\#AS_c}{\#Flw_c}$, where $\#AS_c$ is the number of destination AS and $\#Flw_c$ is the number of flows in c. Values of $\mathcal{R}_{AS_c}$ for P2P applications in our dataset were all found to fit in the interval $[0.3, 0.5]$, whereas values of $\mathcal{R}_{AS_c}$ for other malicious clusters such as scan activities were all found to be less than 0.05. We thus conservatively introduce a minimal threshold $\tau_{as} = 0.2$ for $\mathcal{R}_{AS_c}$, and we discard all netflow clusters c that match the condition $\mathcal{R}_{AS_c} < \tau_{as}$.

4.2 Features Extraction

The use of netflow data for machine learning and botnet detection is often criticized because it provides only generic information such as port numbers or contacted IPs [6]. The raw use of those features usually leads to overfitted models that only detect malware in the initial training set. This paper thus proposes a set of features that goes beyond the intrinsic characteristics of every single netflow record. It better describes the relationship and common trends among all netflow records within a single cluster. Such features capture invariants in C&C channels for P2P botnets. They cannot be easily evaded, yet they are generic enough to detect P2P botnets not initially represented in the training set. Our training features can be grouped into three categories, as follows.

Time-based features: Malware P2P control flows may be similar to benign flows when observed during short intervals of time. However, observing these flows at longer durations may reveal periodicities that are unlikely to exist in benign P2P flows. Table 1 illustrates the periods between communication rounds for P2P malware in our dataset. Time-based features capitalize on this observation in order to characterize the occurrence of control flows within a cluster as a function of time. We leverage periodicities in a cluster using the recurrence period density entropy (RPDE)[15]. RPDE is a normalized metric in time series analysis that determines the periodicity of a signal. It is equal to 0 in case of perfectly periodic signals and equal to 1 for white random noise signals. We compute the RPDE metric, the same as explained in [15], using a time series that represent the flow arrival times within a cluster. In addition to the RPDE, we also compute the mean and standard deviation (std) for inter-flow arrival times in each cluster. The sequence of inter-flow arrival times is derived from the time series by taking the difference between every couple of consecutive flows. However, the mean inter-flow arrival time is a linear metric, as opposed to the RPDE metric that rather applies in the phase space [15].

Space-based features characterize the way a P2P node contacts other peers in the network. P2P bots usually have a lower chunk rate compared to other benign peers [29]. During bootstrap, infected nodes often use hard-encoded lists of peers.

Such lists imply a lower chunk rate, which makes it a distinctive feature for P2P botnets. It is manifested through the number of IP addresses contacted and the port distribution. We characterize space features using the mean and std for the distributions of (new) IP addresses and destination ports contacted. We use the same distribution of destination ports as the one described for P2P filtering, and we add its mean and std to our features vector. Regarding IP addresses, we compute the distributions of both IP addresses and *new* IP addresses contacted. The former represents the distribution of the number of remote IP addresses within n sub-intervals of short duration, compared to the longer duration of a cluster. It characterizes the number of IP addresses concurrently contacted by a P2P node at a given time. The latter distribution, computed the same way as for destination ports, characterizes the chunk rate of a P2P application. We add the means and stds of both distributions to our features vector.

Flow size-based features characterize the number of bytes and packets transferred in P2P flows. They capture specific control operations for a given P2P application [14]. We extract both *unique* and *statistical* flow size features. The former represents the distribution of unique flow sizes against the number of flows that have a given size in a cluster. We compute the mean and std for this distribution and add these to our features vector. On the other hand, statistical flow sizes characterize the regularity of flow size behavior over time within a cluster. We group flows in a cluster into a time series according to their arrival times. We further split this interval into n sub-intervals of equal lengths. For each sub-interval, we compute the mean size for all flows in this interval, thus obtaining a time-based distribution of mean flow sizes in a cluster. We compute both mean and std of this new distribution and add these to our features vector.

4.3 P2P Botnet Detection Model

P2P clusters cannot be all used for training as some of these are likely to appear in both malware and benign P2P flows. In fact, while certain malware implements its own version of P2P protocols (e.g. waledac), others use existing overlay protocols like overnet (e.g. Storm) and kademlia (e.g. TDSS). Clusters provided by the second category of malware may share similar patterns with other benign P2P flow clusters for specific P2P control operations such as keep alive or route discovery, and so they would share similar network footprints. They should be thus discarded prior to building the detection model. In fact, we want to keep only clusters of P2P flows that implement P2P control operations that can be accounted for during detection, such as P2P communication rounds, chunk rates and IP distributions.

We discard non-discriminatory clusters by cross-correlating our combined set of malware and benign P2P clusters. Non-discriminatory clusters include flows triggered by malware and benign P2P applications that use the same P2P protocols (e.g. emule, overnet) and that implement the same P2P functionalities. We apply hierarchical clustering, using our features vector, in order to build a dendrogram where leaf nodes are P2P clusters and the root node is a set of all P2P clusters. Then we use the Davies-Bouldin index [8] to find the best cut in the dendrogram, and so we obtain meta-clusters of malware and benign P2P clusters. A meta-cluster corresponds to a node in the dendrogram, and that includes either or both malware and benign P2P clusters. Discriminatory meta-clusters include almost only malware or benign P2P clusters, and these are kept as input to the training phase. We thus discard as non-discriminatory meta-clusters all meta-clusters where the proportion of malware or benign P2P clusters does not exceed a threshold τ_d. We experimentally set the value of τ_d based on our P2P training set, as seen further in section 5.2.

We tested multiple learning algorithms in order to build our detection model, including SVM, J48 and C4.5 decision tree classifiers [7,22]. SVM provides an extension to nonlinear models that is based on the statistical learning theory. On the other hand, decision trees are a classic way to represent information from a machine learning algorithm, and offer a way to express structures in data. We evaluated the detection rates, including False Positives (FP) and False Negatives (FN), for these available learning algorithms using our labeled set of P2P clusters. We obtained higher detection accuracies using the SVM classifier, and therefore we use this algorithm to build our detection model.

5 Experimentation

This section describes the design of our experiments, as well as the dataset that we used in order to build and validate our approach. First, we build a P2P botnet detection model using a learning set of malware and benign P2P applications. Then we evaluate in this section three properties of our system. First, we use a cross-validation method in order to assess the accuracy of our P2P botnet detection model. Then, we evaluate the contribution for the different features of our model towards detection and we discuss results of these experiments using our initial P2P learning set. Last of all, we evaluate the coverage of our system through application to live netflow traffic.

5.1 Training Dataset

We obtained samples of malware traffic from a security company which collects binaries using its own honeypot platform. Traffic samples, provided as pcap files labeled with the malware md5, included 1 hour of malware network activity collected during execution in a sandbox. We use the virustotal API in order to identify P2P malware samples in our dataset using their md5 labels. We extracted all malware samples that were matching with more than 10 known P2P antivirus signatures in virustotal. In fact current antivirus scanners usually provide conflicting signatures for a same malware sample. Yet we need a valid ground truth of P2P malware samples in order to build and evaluate our system. Hence, we keep only malware samples that match with more than a single known P2P malware signature in virusTotal. We empirically set the number of matching P2P signatures to 10, which is almost the third of matching antivirus

Table 1. Malware samples by families of malware

Family	P2P functionality	Period	P2P Protocol
Sality (v3, v4)	Primary C&C	40 min	Custom protocol, encrypted-RC4 packets over UDP
Kelihos	Primary C&C	10 min	Custom protocol, P2P over HTTP traffic
Zeus v3	Primary C&C	30 min	Kademlia-like protocol transported over UDP
TDSS	Backup C&C	–	Public Kad network
Waledac	Primary C&C	30 sec	Custom protocol, P2P over HTTP traffic
ZeroAccess v1	Primary C&C	15 min	Custom protocol, transported over TCP and UDP
Storm	Primary C&C	10 min	Overnet protocol

scanners, and that we believe it provides enough confidence of a P2P malware sample. We obtained an overall number of $1,317$ distinct malware samples. Note that malware uses P2P protocols for different purposes, including primary C&C, bootstrap, spreading and failover. Table 1 illustrates the most predominant malware families that we found in our dataset.

We process malware samples using our P2P filter in order to build clusters of P2P flows. We discard non-P2P flows using our coarse filter, then we create clusters of flows for each malware sample. We implement the incremental k-means algorithm, using different values for the threshold τ_{cl}. A low value of τ_{cl} causes the clustering process to create almost a new cluster for every new flow. Such clusters are usually discarded by the fine filter because of their relatively low AS and port distributions, thus leading to higher false negatives rates. On the other hand, a high value for τ_{cl} regroups flows that are not similar within the same clusters. These clusters would include flows that are not triggered by the same application, thus leading to more false positives. To find the best trade-off for τ_{cl}, we applied the clustering process to our training set of P2P malware. We tested several values for τ_{cl}, each time using the output clusters as input to our fine P2P filter. The optimal value for τ_{cl} is the highest value that still provides zero false negatives, assuming that there would be no false positives in case of only P2P traffic. In our case, we achieved a maximum detection accuracy for a value of $\tau_{cl} = 60$. We obtained around 10 thousand clusters, that is almost 10 netflow clusters for each malware sample. The fine P2P filter further provided around 3 thousand clusters, other clusters being dropped as non-P2P flows. In fact, we observed that almost 60% of discarded clusters include less than 40 flows. These clusters mostly implement network discovery protocols or activities related to the sandbox environment (e.g. SMB service). We also observed among discarded clusters a long tail of clusters that include a high number of flows. These are all scan clusters that were mostly discarded by the AS-based filter. The remaining clusters include mostly P2P flows and spam flows. The latter were discarded by the port-based filter, thus obtaining a total number of $2,975$ malware P2P clusters.

On the other hand, we build benign P2P clusters using traffic that we collected from a well protected corporate network. It consists of netflow packets obtained during one month of activity for nearly 150 network terminals. Unfortunately certain P2P applications such as bittorrent were banned due to policy restrictions. We thus completed our training set by manually executing P2P applications in a controlled environment. Then we applied our filter in order to build clusters of benign P2P flows. We obtained a total number of 415 benign P2P clusters, associated with 53 distinct IP addresses. Almost half of our benign P2P clusters included skype flows (230), but we also obtained other clusters such as eMule (43), kademlia (37) and Gnutella (35). We mostly obtained skype flows mainly because the corporate network is aimed for professional usage, and other P2P applications were being occasionally used. On the other hand, we collected 379 benign P2P clusters from manually executed P2P applications, including bittorrent, eDonkey and Manolito. Our training set thus included 794 benign P2P clusters that we used as input to our detection model.

5.2 P2P Botnet Detection Model

The P2P filter provided a resulting set of 3, 769 P2P clusters, including 2, 975 of malware and 794 of benign P2P clusters. The purging process discards P2P clusters that are shared between both malware and benign P2P flows. Hence, we cross-correlated our set of 3, 769 P2P clusters, using hiearchical clustering, and we obtained 53 meta-clusters, including 41 discriminatory meta-clusters. The latter include more than 93% of only benign or malware P2P flows, such as 7 meta-clusters which clearly included Sality flows, 4 meta-clusters included Waledac, and 9 meta-clusters included Skype flows. On the other hand, 12 meta-clusters included both malware and benign flows, and so they are discarded as non-discriminatory clusters. For instance, and regarding the kademlia protocol, 10 meta-clusters were found to include kademlia-like P2P clusters. In fact kademlia protocol includes 4 message types: `ping`, `store`, `find_node` and `find_value`. 7 meta-clusters included more than 93% of only malware or benign P2P flows. These meta-clusters included strictly `find_node` and `find_value` messages. Malware and benign P2P clusters were falling into different meta-clusters mostly because of their different chunk rates. The three remaining meta-clusters included between 60 and 70% of malware clusters. These clusters included mainly `ping` requests, which are dropped by the purging module as being non-discriminatory flows. We obtained as output to this process a training set of 2, 647 P2P clusters, including 2, 143 malware clusters and 504 benign P2P clusters.

Cross-Validation: We performed a cross validation experiment, using our labeled ground truth dataset, in order to evaluate the detection accuracy of BotSuer. We split our malware dataset into two subsets: 80% of malware samples that we use for training, and the remaining 20% that we use for evaluation. Yet for the 53 IP addresses that were using P2P protocols in the corporate network, we randomly extracted 10 IP addresses from our training set so we can use use them for evaluation. Then we merged traffic for our 20% malware evaluation set with random IP addresses that we extracted from the corporate network traffic. We further use our training set of malware and benign P2P traffic as input to our P2P filter, and then we applied our cluster purging module with different values of the threshold τ_d. We obtained for each value of τ_d a different number of labeled P2P clusters that we use to train our SVM classifier. We evaluated the

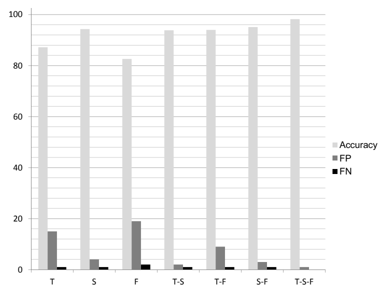

Fig. 2. Contribution of features towards detection

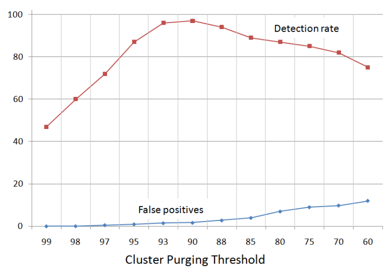

Fig. 3. Detection rate and false positives rate for BotSuer

detection models that we obtained using the validation dataset, including traffic from the corporate network merged with the traffic from the 20% remaining malware samples. We measured the detection rate and the false positives rate for each value of τ_d, and that we illustrate with the ROC curve in Fig. 3.

As in Fig. 3, a high value for τ_d - *i.e. closer to the y-axis* - leads to lower detection rates and less false positives. In fact, a high value for τ_d discards more clusters during the learning process, and so it reduces the coverage of BotSuer. On the other hand, lower values for τ_d allows less discriminatory clusters to go through the purging process. These would reduce the accuracy of BotSuer, leading to a higher false positives rate and a lower detection accuracy. We found a linear increase in the detection rates for values of τ_d lower than 93%. Yet we obtained the best detection rates for values of τ_d in the interval $[90 - 93\%]$, including 97% detection rate and 1.6% false positives.

Contribution of Features towards Detection: We used the cross-validation method to evaluate the contribution of our features towards detection. We built detection models by separately using each class or combinations of these classes, and then we evaluated the detection accuracy, including false positives and negatives, as illustrated in Fig. 2.

Our model achieves almost 97% detection accuracy when combining all classes of features. When evaluated separately, space-based features ($\mathcal{S}$) provided the best detection accuracy (93%). We believe this is due to the fact that our learning set includes netflow packets collected during only 1 hour of malware execution time. Hence, time-based features provided a lower accuracy because the execution time is not long enough to accurately characterize periodicities in P2P control flows. On the other hand, size-based features provided low detection accuracy when solely used to build the detection model, almost with 20% false positives rate. This was not surprising because malware may still bypass size-based features by adding noise or paddings to P2P control flows. Although only few malware currently uses such techniques, we observe that size-based features cannot be used as standalone features for P2P botnet detection. As opposed to time and space features, size-based features may still be bypassed without modifying the underlying overlay protocol.

Testing with ISP Flows: The test against ISP flows was indeed challenging because we lack the ground truth about the nature of detected infections. We trained our detection model with all P2P clusters at our disposal, and we used the value of τ_d that provided the best detection accuracy. The ISP flows included 3 hours of anonymized netflow for almost $4,347$ distinct IP addresses, collected at a peek traffic rate. The DNS coarse filter was applied at the source, and so the netflow trace that we obtained included only flows not preceded with susccessful DNS resolutions. We split this traffic into one hour length intervals, which corresponds to the malware execution time in our dataset. Then we applied our P2P filter and botnet detection model on traffic in each time interval. Our filter extracted 793 P2P flow clusters, associated with 146 distinct IP addresses. In order to validate our P2P filter, we also obtained from the ISP a list of the anonymized IP addresses that were found to be implementing P2P applications in our netflow trace, and that were detected using proprietary P2P protocol signatures. We admit that these signatures cannot formally validate our approach, but they may still provide a ground truth to evaluate our results. P2P signatures detected 169 distinct IP addresses that implement P2P applications, including 23 IP addresses not detected by our filter. In fact 18 of these addresses triggered less than 10 P2P flows. They provided small P2P clusters that are discarded by our P2P filter. We asked the ISP to verify about the origin of these flows since the source IP addresses for the traffic at our disposal were all anonymized. We found that these were mostly signaling flows for external IP addresses being routed through the ISP network. Therefore, we would not consider them as false negatives. On the other hand, the 5 remaining IP addresses detected by the P2P signatures were indeed false negatives. They mostly included utorrent P2P over HTTP flows, and so they were discarded by our port-based filter. They were also discarded by the AS-based filter most likely because these P2P applications were dormant during the observation window. In fact we observed mostly incoming flows, but there were relatively little outgoing P2P flows for these IP addresses.

We used the 793 P2P clusters as input to our detection model. It identified 11 malicious flow clusters associated with 3 distinct IP addresses. Since traffic was all anonymized, we validated our approach using public IP blacklists[2]. In fact we consider a cluster to include malware P2P flows when at least 20% of remote IP addresses in this cluster appear in public backlists. Indeed we identified using these blacklists 4 netflow clusters as being malicious, *all associated with the same IP address*. We thus consider this as a strong evidence of a malware infection, and so it is a true positive. Yet the same infected IP address was appearing in the list of IP addresses that were found to be implementing P2P applications by the proprietary ISP signatures. This is another clear evidence of the ability of BotSuer to detect P2P bots that use known benign P2P protocols. Unfortunately we couldn't validate the 7 remaining clusters using the publicly available blacklists, and so they are likely to be misclassified by our system. We thus achieved 0.8% false positives, associated with two distinct IP addresses

[2] RBLS is a free API to check multiple public IP blacklists - `http://www.rbls.org/`

during 3 hours of traffic monitoring for 4, 347 distinct IP addresses, which is a fairly reasonable number of alerts to be handled by the system administrator.

6 Discussion

BotSuer detects malware using statistical features such as chunk rates, periodicities and botnet distribution. It also classifies flows for a specific P2P application using high level features such as flow size and number of packets. It would be thus unable to accurately detect P2P bots if the attacker modifies P2P communication intervals, contacts a larger set of peers, or uses random paddings in its malware P2P traffic. Such maneuvers modify the statistical consistency in malware P2P flows and so it makes detection more difficult. However, these techniques require an attacker to rebuild its malware P2P toolkit. They also increase overhead and reduce botnet stability, which makes botnet management more difficult. Indeed botnets would no longer be able to dissimulate within benign P2P flows, and so they will be exposed to other detection techniques.

On the other hand, BotSuer differentiates malware and benign P2P control flows using only a binary classification. We aim to extend this study by proposing new techniques to identify specific botnet families during detection. We will explore unsupervised clustering algorithms that apply to our set of malicious P2P flows. These algorithms separate in different clusters P2P flows that are likely to be generated by common families of botnets (e.g. Sality, Storm, Nugache).

Certain malware uses P2P protocols only as a failover mechanism to replace its primary C&C channels, and so it would not contribute to building the detection model. In [17], authors propose an approach that detects primary C&C channels during malware execution in a sandbox. It dynamically intercepts primary C&C channels and forces malware to engage in a failover strategy. Using techniques such as [17] enables to trigger P2P failover strategies so we can take these into account in our detection model. These techniques usually apply during malware sandbox analysis and so they are out of scope in this study.

7 Conclusion

This paper presented BotSuer, a new approach to detect P2P bots inside a network perimeter. To the best of our knowledge, BotSuer is the first to detect even single infected P2P bots. BotSuer implements a fully behavioral approach to detect malware infected nodes in the network. Yet it does not use deep packet inspection nor intrusion detection alerts. It is thus resilient against malware obfuscation mechanisms and detects bots that use encrypted P2P communications for command and control. It also detects stealthy P2P bots, as well as targeted infections inside a network perimeter. We tested BotSuer against real world P2P traffic, including malware and benign P2P flows. Our experimental results validate our approach, which provides a high detection accuracy with a very low false positives rate.

References

1. Cuckoo: Automated malware analysis system (2010), `http://www.cuckoobox.org/`
2. Anubis: Analyzing unknown binaries (2011), `http://anubis.iseclab.org`
3. Aberer, K., Hauswirth, M.: An overview on peer-to-peer information systems. In: Proceedings of the 4th workshop on Distributed Data and Structures (2002)
4. Bilge, L., Balzarotti, D., Robertson, W., Kirda, E., Kruegel, C.: Disclosure: Detecting botnet command and control servers through large-scale netflow analysis. In: Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC (2012)
5. Bilge, L., Kirda, E., Kruegel, C., Balduzzi, M.: Exposure: Finding malicious domains using passive dns analysis. In: Proceedings of the 18th Network and Distributed System Security Symposium, NDSS (2011)
6. Claise, B.: Cisco systems netflow services export version 9. RFC 3954 (October 2004)
7. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods. Cambridge University Press (2000)
8. Davies, D.I., Bouldin, D.W.: A cluster seperation measure. IEEE Transactions on Pattern Analysis and Machine Intelligence (1979)
9. François, J., Wang, S., State, R., Engel, T.: BotTrack: Tracking botnets using netFlow and pageRank. In: Domingo-Pascual, J., Manzoni, P., Palazzo, S., Pont, A., Scoglio, C. (eds.) NETWORKING 2011, Part I. LNCS, vol. 6640, pp. 1–14. Springer, Heidelberg (2011)
10. Grizzard, J.B., Sharma, V., Nunnery, C., Kang, B.B.: Peer-to-peer botnets: Overview and case study. In: Proceedings of USENIX HotBots (2007)
11. Gu, G., Perdisci, R., Zhang, J., Lee, W.: Botminer: Clustering analysis of network traffic for protocol and structure independent botnet detection. In: Proceedings of the IEEE Symposium on Security and Privacy, SSP (2008)
12. Kapoor, A., Mathur, R.: Predicting the future of stealth attacks. In: Virus Bulletin (2011)
13. Karagiannis, T., Broido, A., Brownlee, N., Claffy, K., Faloutsos, M.: Is p2p dying or just hiding? In: IEEE GLOBECOM, vol. 3, pp. 1532–1538 (2004)
14. Karagiannis, T., Broido, A., Brownlee, N., Claffy, K., Faloutsos, M.: File-sharing in the internet: A characterization of p2p traffic in the backbone. UC Riverside technical report (November 2003)
15. Little, M.A., McSharry, P.E., Roberts, S.J., Costello, D.A., Moroz, I.M.: Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection. Biomedical Engineering Online 6 (2007)
16. Nagaraja, S., Mittal, P., Hong, C.-Y., Caesar, M., Borisov, N.: Botgrep: Finding p2p bots with structured graph analysis. In: Proceedings of the 19th USENIX Security (2010)
17. Neugschwandtner, M., Comparetti, P.M., Platzer, C.: Detecting malware's failover c&c strategies with squeeze. In: Proceedings of the 27th Annual Computer Security Applications Conference, ACSAC (2011)
18. O'Kane, P., Sezer, S., McLaughlin, K.: Obfuscation: The hidden malware. In: IEEE Security & Privacy, pp. 41–47 (2011)
19. Ollmann, G.: Botnet communication topologies: Understanding the intricacies of botnet command-and-control. Damballa White Paper (2009)
20. Ordonez, C.: Clustering binary data streams with k-means. In: Proceedings of the 8th Workshop on Research Issues in Data Mining and Knowledge Discovery, pp. 12–19 (2003)

21. Porras, P., Saidi, H., Yegneswaran, V.: Conficker c p2p protocol and implementation. Technical report, Computer Science Laboratory, SRI International (2009)
22. Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers (1993)
23. Stover, S., Dittrich, D., Hernandez, J., Dietrich, S.: Analysis of the storm and nugache trojans: P2p is here. In: USENIX, vol. 32 (December 2007)
24. Stutzbach, D., Rejaie, R.: Understanding churn in peer-to-peer networks. In: Proc. ACM SigComm Internet Measurement Conference (2006)
25. Symantec. Internet security threat report. 2012 Trends 18 (April 2013)
26. Tenebro, G.: W32.waledac threat analysis. Symantec Technical Report (2009)
27. Trusteer. No silver bullet: 8 ways malware defeats strong security controls (2012), Whitepaper accessible on `http://www.trusteer.com/resources/white-papers`
28. Willems, C., Holz, T., Freiling, F.: Cwsandbox: Towards automated dynamic binary analysis. In: IEEE Security & Privacy (2007)
29. Yen, T.-F., Reiter, M.K.: Are your hosts trading or plotting? Telling p2p file-sharing and bots apart. In: 30th Conf. Distributed Computing Systems (2010)
30. Zhang, J., Perdisci, R., Lee, W., Sarfraz, U., Luo, X.: Detecting stealthy p2p botnet using statistical traffic fingerprints. In: Proc. 41st DSN (2011)