Behavioral fine-grained detection and classification of P2P bots

Nizar Kheir[a], Xiao Han[a], Chirine Wolley[b]

[a] *Orange Labs, France*
[b] *Université d'Aix-Marseille, France*

Abstract

Modern botnets are increasingly shifting towards overlay networks, using peer-to-peer (P2P) protocols, for command and control (C&C). P2P botnets are robust against detection and takedown as they avoid single nodes of failure, and mostly use custom encrypted C&C communications. Pattern-based signatures are also inappropriate, yet they cannot efficiently detect malware that uses benign P2P applications such as Kademlia and Overnet. This paper presents PeerMinor, a fully behavioral system that detects and classifies P2P bots inside corporate networks. PeerMinor learns the behavior of known malware and benign P2P applications in order to detect P2P bots and provide security administrators with a correct diagnosis of ongoing malware infections.

PeerMinor operates in two phases, learning and detection. In the learning phase, it processes known malware and benign P2P traffic in order to build a two-stage classifier. In the first stage, PeerMinor uses supervised learning in order to build a detection model that separates malicious and benign P2P network activity. In the second stage, it builds a one-class classifier for each known P2P malware family, and uses these classifiers to associate detected P2P bots with a known malware family where possible, thus providing a better situational awareness for system administrators. During detection, PeerMinor processes network traffic using its learning-based model in order to detect P2P bots. To the best of our knowledge, PeerMinor is the first behavioral system that goes beyond simple detection in order to provide an accurate diagnosis about ongoing malware infections. Experimental results prove that PeerMinor achieves both scalability and accuracy. It uses only network features with no need of pattern-based signatures, which can be easily evaded by botnet herders.

Keywords: P2P botnets, Behavioral detection, Machine learning, netflow, intrusion detection

Email addresses: nizar.kheir@orange.com (Nizar Kheir), xiao.han@orange.com (Xiao Han), chirine.wolley@etu.univ-amu.fr (Chirine Wolley)

1. Introduction

Botnets have become by far the spearhead of malware activity on the Internet today. They constitute networks of infected nodes that communicate with an attacker through a common Command and Control (C&C) infrastructure. Botnet topologies have been constantly evolving in the recent years, yet current detection systems can no longer keep pace with their growing threat [1]. In fact, botnets are no longer being used only to trigger massive distributed attacks such as spam and DDoS, but more often to seek financial benefits and sensitive user data [2, 3]. They only trigger little network traffic, mostly signaling flows that are usually encrypted in order to evade network IDS signatures. Botnet herders also use code obfuscation and binary polymorphism in order to hide malware payloads so they get unnoticed by anti-virus solutions [4, 5]. Network monitoring systems are thus being confronted to stealthy botnets, along with encrypted communications and only few infections inside a network perimeter [6].

Indeed, the recent years have witnessed a major shift in botnet C&C activity, increasingly relying on distributed overlay networks that add more flexibility and resilience to botnet topologies [7]. Hence, malware started developing peer-to-peer (P2P) capabilities, using either benign-like P2P applications such as Kademlia (e.g. the TDSS botnet) and Overnet (e.g. the Storm botnet), or custom P2P protocols such as Waledac and Zeus [8]. Although being difficult to setup and manage, P2P botnets are robust as they dispose of single nodes of failure in their architectures. Yet detection and classification of P2P botnets add more challenges because of the following reasons.

First of all, P2P traffic is difficult to characterize using only network layer features. In fact, benign P2P applications increasingly use random ports and encrypted payloads, so they can no longer be detected using protocol-based signatures [9, 10]. Therefore, P2P botnets can easily dissimulate their signaling activity as part of benign P2P communications [11].

Second, P2P bots cannot be detected using blacklists of domain names or IP addresses, as for centralized botnet topologies [12]. In fact, P2P bots locate and retrieve commands using the overlay network. Two infected bots are unlikely to connect to the same set of peers, even though some overlaps may occur. Therefore, detection of P2P infected nodes should be based on the way these nodes interact with the overlay network, and not based on the single IP addresses or domain names being contacted.

Last of all, persistent malware is becoming stealthier in the way it spreads infection over the network. Security administrators are oftentimes confronted with a limited number, yet a single infection by the same malware inside their network perimeter. Hence, botnet detection within the same administrative domain is becoming increasingly difficult when relying only on botnet swarm effects [13]. In fact current botnet detection systems must be able to detect even a single infection inside the domain to be protected.

This paper builds upon previous work for the same authors in [26, 27] and proposes a behavioral, scalable and fine-grained P2P bot detection system. In [27], authors propose a behavioral malware classification system that is designed

to work offline. It processes network communications for malware running in a sandbox and associates this malware with a known malware family. The system in [27] cannot differentiate flows for malicious and benign P2P applications that are running on the same network terminal. Hence, this system cannot be easily integrated within an inline P2P botnet detection system. On the other hand, authors propose in [26] a system that detects active P2P bots through network analysis. It tells apart malicious and benign P2P flows for a given network terminal, but cannot associate infected terminals with a given malware family. Therefore, the main contribution of this paper is a new architecture that integrates both malware detection [26] and classification [27] functionalities in a more comprehensive P2P botnet detection system.

The main idea in this paper is that malware belonging to the same family embeds the same P2P toolkits and communicates with the overlay P2P botnet in the same way. In fact P2P protocols include both control flows and data transfer flows [14]. Transfer flows mostly depend on the data being shared through the network. However, control flows carry signaling activities such as node discovery, route requests, search requests and keep-alive, and that reliably characterize the P2P protocol being implemented. As shown in [15], 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 have similar network behavior in terms of flow size, number of packets and flow duration. While certain clusters, such as keep-alive flows, may be common for both malware and benign P2P applications, 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 [2]. Yet the Sality botnet has a lower chunk rate for route discovery requests, compared to other benign P2P applications.

Our system, called PeerMinor, observes network traffic for known malware samples and uses several heuristics to isolate relevant P2P signaling activities. It further builds a behavioral detection model using only network-layer features, with no need for deep packet inspection. PeerMinor describes the network activity of a P2P malware using high level features such as periodicities, chunk rate and geographical distribution. It operates in two phases, the learning phase and the detection phase. During the learning phase, it first builds a supervised learning system that is used to inspect P2P traffic and extract botnet C&C activity. Then it implements unsupervised clustering mechanisms in order to build families of P2P malware and uses these families to associate a detected bot infection with a known P2P malware family where possible.

During detection, PeerMinor processes network traffic and builds clusters of P2P flows for each network terminal. It uses P2P flow clusters in order to build a network footprint for each terminal, and matches this footprint against its malware P2P classifiers in order to detect and characterize infected P2P bots.

The main contribution of this paper, compared to the previous work for the same authors in [26, 27], is scalability. We achieve scalability by simplifying the multi-step P2P filtering process proposed in [26] and leveraging incremental clustering algorithms that run efficiently on very large datasets. They apply

first on an IP basis in order to discard non P2P flows triggered by each single endpoint, and then regroup and correlate P2P flows in order to build network footprints that can be compared against known malware families. We would like to emphasize that scalability is an important requirement for the system described in [27], and that we introduce in this paper. In fact building network footprints for every single endpoint is a greedy process as it requires two rounds of flow clustering and filtering applied to every single endpoint. Therefore, the architecture that we propose in this paper incorporates two functionalities, including first a P2P inspection module that rapidly discards endpoints that do not run any P2P application, as well as endpoints that are likely to be acting as benign P2P nodes. Only infected P2P nodes are handed for malware family recognition by the classifier module. To summarize, PeerMinor both detects P2P bots and associates them with appropriate P2P botnet families, if any. To the best of our knowledge, it is the first to combine behavioral detection with subsequent diagnosis and malware family identification. Indeed, our experimental results prove the ability of PeerMinor to accurately detect and classify P2P bots with a very low false positives rate.

The remaining of this paper will be structured as follows. Section 2 describes related work. Section 3 presents the architecture and workflow of our system. Section 4 develops our experiments and the process that we used to validate our system. Section 5 discusses techniques to evade our system and shows how PeerMinor is resilient against these techniques. Finally, section 6 concludes.

2. Related work

Related work includes several techniques to detect and classify botnets through behavioral traffic analysis. They usually provide a binary classification of P2P nodes, being either infected or benign [13, 16, 17, 19, 20, 21, 22]. Yet there is only few approaches that build families of P2P malware based on its network behavior, and use these families during detection and diagnosis [23, 24].

Solutions such as BotGrep [13], BotTrack [19] and BotMiner [20] offer similar techniques to correlate netflow data [25] and localize P2P bots based on their overlay C&C topologies. They build clusters of hosts based on their network activity and isolate groups of hosts that constitute common P2P networks. They further separate malicious and benign P2P groups by correlating information from multiple sources such as honeypots and intrusion detection systems. Unfortunately, these techniques are inappropriate against modern botnets. In fact botnets are becoming stealthier and so they cannot be easily detected using only IDS signatures, thus limiting the coverage of these techniques.

Bilge et al. propose an alternative approach that detects botnets using large scale netflow analysis [17]. They observe ISP traffic and detect botnets through the coordinated activity for groups of infected nodes. This approach detects only centralized botnet architectures, and cannot accurately detect distributed P2P botnets. As opposed to [17], other existing studies offer to detect infected P2P bots only inside a given network perimeter [21, 22]. They first discard non-P2P traffic using heuristics such as DNS traffic and failed connection attempts.

Then they build groups of P2P nodes that have the same network behavior or that connect to similar IP addresses. They further propose a similarity degree in order to detect nodes that are likely to be part of a same botnet. These studies can only detect P2P bots when multiple infected nodes belong to the same botnet inside a network perimeter. Yet they only provide a binary classification, with no ability to identify common malware families.

Another trend of research consists of passively observing DNS traffic in order to detect malicious C&C domains [16, 18]. Authors in [18] build a dynamic DNS reputation system that uses both network and zone features of a given domain. They consider that the malicious use of DNS has unique characteristics and can be separated from benign, professionally provisioned DNS services. Therefore, they passively observe DNS queries and build models of known benign and malicious domains. These models are used to compute a reputation score for a newly observed domain, and which indicates whether this domain is indeed malicious or benign. On the other hand, Bilge et al. in [16] propose an alternative approach that applies machine learning to a set of 15 DNS features in order to identify malicious C&C domains. They build a learning set of known benign and malicious domain names that is used to train a DNS classifier. This classifier passively monitors real-time DNS traffic and identifies malware domains that do not appear in existing blacklists. Features in [16] are grouped into four categories, including time-based features, response features, TTL features and syntactical domain name features. They characterize anomalies in the way a given domain name is being requested, including abrupt changes in DNS queries towards this domain. Unfortunately, P2P botnets constitute overlay networks where bots communicate directly using IP address in the overlay network, with no need to use the DNS service. Hence, detection of botnets using DNS features cannot be efficiently applied to detect P2P bots.

On the other hand, PeerRush provides in [23] a similar approach to our system that aims at classifying P2P flows and identifying specific P2P protocols or applications. It uses features such as inter-packet delays and flow duration in order to classify P2P applications. PeerRush achieves good detection accuracies against benign P2P applications. However, it is not clear how this system will contribute to classifying P2P botnets. For example, inter-packet delays can be easily evaded and these are weak indicators of P2P activity. Yet PeerRush deals with all P2P signaling flows as a whole. It does not classify flows according to their embedded message types and the rate of each signaling activity. This paper provides a better alternative to PeerRush, by offering to separate detection and diagnosis of P2P malware infections. In fact our system includes a P2P inspector module that first tells apart malicious and benign P2P applications. Hence, only network footprints for malicious P2P nodes are used in our system as input to a malware classifier module, and that associates these footprints with known P2P malware families where possible.

Last of all, Hu et al. [24] use flow statistics to build behavior profiles for P2P applications. They experimented only with two P2P applications (PPLive and BitTorrent), and did not consider malicious P2P activity. Yet, they do not separate P2P control and data traffic. In fact data flows do not clearly

characterize P2P botnet C&C activity as they depend on the content being shared. Therefore, we classify in this paper P2P signaling flows and use only these as a basis for P2P botnet detection. To the best of our knowledge, our approach is the first to combine both behavioral detection and diagnosis in order to detect and correctly characterize ongoing malware infections inside a given network perimeter.

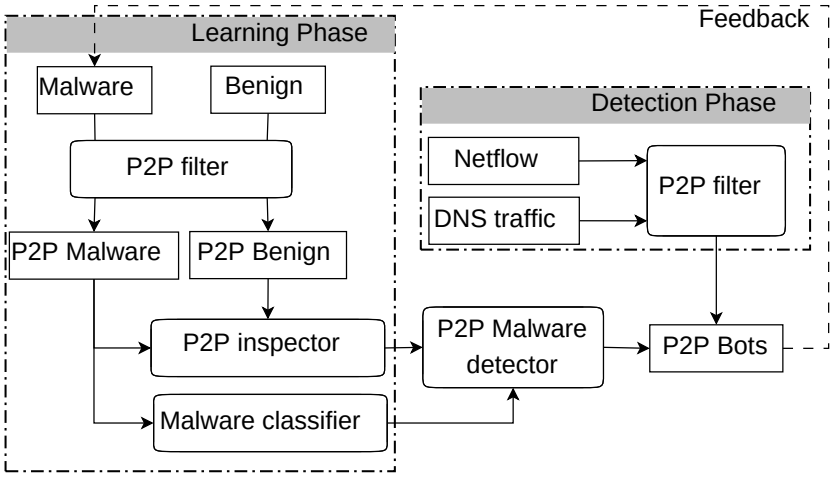


Figure 1: Botnet Detection System

3. Botnet detection system

3.1. System overview

PeerMinor detects and classifies P2P bots using only network traffic features. As in figure 1, it builds a P2P malware detection model using a learning set of malware and benign P2P traffic. Then it applies this model to network traffic in order to detect P2P bots inside a given network perimeter. PeerMinor goes beyond the simple detection of P2P bots in order to associate infected nodes with a known P2P malware family where possible. As illustrated in figure 1, the training phase of PeerMinor includes two main components: the P2P inspector and P2P malware classifier. P2P inspector uses supervised learning to separate malicious and benign P2P traffic when observed for a given network node. On the other hand, P2P malware classifier builds a P2P footprint for every *infected node* and assigns this footprint to a known P2P malware family. PeerMinor also notifies the administrator of a new, yet unknown malware P2P footprint so it can be submitted to a deeper manual analysis. While the P2P inspector and malware classifier modules were separately introduced in [26] and [27], PeerMinor offers a framework that combines both functionalities in a more comprehensive P2P botnet detection system.

3.1.1. P2P filter module

During training, PeerMinor processes malware traffic obtained through execution in a sandbox environment (e.g. Anubis [28], CWSandbox [29] or Cuckoobox[1]). Identification and extraction of P2P malware samples from within a malware database can be performed using either a ground truth malware dataset such as the VirusTotal API[2], or directly using the behavioral P2P filter module implemented by PeerMinor. To the purpose of this paper, and in order to build a ground truth dataset to validate our system, we pick-up P2P malware using antivirus (AV) signatures in virusTotal. Nonetheless, P2P malware also triggers non-P2P flows in addition to its main C&C communications. Therefore, PeerMinor implements a P2P filter that uses several heuristics to discard flows that do not carry any P2P activity during analysis. For instance, the rate of failed connection attempts is usually used as a way to detect P2P applications. Therefore, our filter discards malware whose rate of failed connection attempts does not exceed a given threshold. It also uses other heuristics such as flows initiated after successful DNS requests, number and geographical distribution of remote contacted IPs. Our experimental results prove that our filter is indeed effective in eliminating non-P2P malware, with almost no false positives.

As opposed to P2P malware, benign P2P flows are difficult to collect as we often lack the ground truth about the legitimate behavior of P2P applications on the network. Indeed, executing benign P2P applications in a controlled environment does not reveal all intricacies of real P2P traffic. In this paper, we collect netflow packets from a well-protected corporate network. Terminals in this network abide to strict access policies. They are all equipped with an updated AV solution and outbound traffic from this network is monitored using a proxy server with SSL inspection capability. Hence, we can reasonably consider as benign all traffic collected from this network. Although we cannot discard the possibility of few terminals being infected, these are limited compared to the overall amount of traffic and so they would have little impact on our system. As in figure 1, PeerMinor discards non-P2P flows for this network and keeps only benign P2P flows the same way as it did for malware traffic.

3.1.2. P2P inspector module

The P2P inspector builds a detection model using the labeled set of malware and benign P2P flows. First, it groups P2P flows for each malware or benign P2P application into clusters where they are similar enough so they are likely to implement the same P2P functionality (P2P keep alive, P2P search queries, P2P route discovery). It uses for this purpose only network layer features such as flow size, number of packets, and bit rates. In fact control flows that implement the same P2P signaling activity for the same P2P protocol have similar network features and so they will be grouped within same clusters.

The P2P inspector further characterizes the P2P activity for flows in a given

[1] `http://www.cuckoobox.org/`
[2] `http://www.virustotal.com/`

cluster using a comprehensive set of features, including time, space and flow size-based features. Time features capture unusual sequences of flows and periodicities in a cluster. Space features capture the dynamics of a P2P network such as geographical distribution and chunk rate, including the rate of (new) IP addresses and ports contacted by a P2P application. Size-based features capture high-level properties such as the bit rates and packets shared by flows in a cluster. The P2P inspector uses these features in order to first prune the training set and eliminate non-discriminatory P2P clusters. It cross-correlates P2P clusters and discards those that have similar properties in both malware and benign P2P applications. These are mostly clusters of P2P keep-alive flows that are likely to have similar implementations in both categories of P2P application. The inspector module uses the remaining discriminatory clusters as input to train and build the P2P botnet detection model. By grouping flows into clusters where they implement the same P2P functionality, the P2P inspector detects infected P2P bots even when they execute benign P2P applications. In fact, malicious and benign P2P flows triggered by the same terminal will be grouped into different clusters as they have different P2P characteristics. Therefore, malicious P2P flows will be grouped into different clusters and so they will be correctly detected by the inspector.

3.1.3. P2P classifier module

During training, the P2P classifier processes only malware P2P flows. It groups together samples of the same malware family, and builds a one-class classifier for each P2P malware family. During detection, malicious P2P flows detected by the inspector module are used by the classifier in order to associate these with a known malware family where possible.

Malware samples that belong to the same family have similar P2P signaling activities, and therefore their P2P flows share similar high level features. As opposed to the inspector module that builds clusters of flows for every single malware sample, the classifier module builds clusters of flows by comparing P2P flows for all malware in the initial learning set. It uses unsupervised clustering in order to group together similar malware flows that are likely to implement the same P2P activity. The flow clustering process uses high-level malware traffic features such as flow size, number of packets, bits per packet, and flow duration. It provides a multiple set of clusters, each one including P2P flows triggered by multiple malware samples, but carrying the same P2P signaling activity (e.g. keep-alive, route discovery, search request, push data) and protocol.

Malware of the same family has its P2P flows grouped within the same clusters because they carry the same P2P signaling activities. The malware classifier module uses P2P flow clusters in order to build families of malware that implement the same P2P protocol. It builds a P2P footprint $\mathcal{F}_\alpha\{c_i\}_{i=1}^m$ of size m for each malware $\mathcal{M}_\alpha$, and which specifies the rate of malware P2P flows within each cluster $\{c_i\}_{i=1}^m$. In other terms, the feature $\mathcal{F}_\alpha\{c_k\}$ would be set to 0 if malware $\mathcal{M}_\alpha$ has no flows in cluster c_k, and it will be set to 1 if it has all its P2P flows in c_k. PeerMinor uses malware footprints as a training set to build P2P malware clusters, each cluster representing a new P2P malware

family. Hence, malware that is grouped by PeerMinor into the same family implements the same P2P protocol and has the same P2P botnet topology.

3.1.4. P2P detector module

The P2P detector applies both inspection and classification models to network traffic in order to detect P2P bots, with no need for deep packet inspection. The P2P inspection module provides a single classifier that tells apart malicious and benign P2P nodes. It is used by the detector module in order to rapidly discard nodes that are unlikely to implement botnet-related P2P activity. Remaining nodes are further provided as input to the one-class classifiers generated by the classifier module. These classifiers are used by the detector in order to associate an infected node with a known P2P malware family where possible. Our system thus provides the administrator with appropriate information in order to characterize and further remediate ongoing P2P malware infections. Infected nodes that are unknown to the classifier module are identified by our system as new P2P malware. These can be further submitted to the security administrator for a deeper manual analysis.

3.2. System description

3.2.1. P2P flow filtering

PeerMinor implements several heuristics to discard flows that do not carry P2P signaling activity. In fact P2P traffic has multiple properties that are different from other centralized network communications, and that can be characterized using high-level network behaviors. This paragraph describes properties that we use in order to discard non-P2P traffic and keep only P2P flows for malware detection and diagnosis. A malware P2P flow thus represents a network connection triggered by malware, and that carries signaling activity associated with the P2P botnet communication. A formal definition of malware and benign P2P flows is further provided in section 3.2.2.

DNS filtering is commonly used to discard non-P2P traffic. Nodes in a P2P network operate outside the DNS system [9]. They search for other peers using routing tables in the overlay network, without prior DNS requests. Although access to a central server through DNS resolution is possible at bootstrap, nodes further communicate directly using IP addresses, and access to the DNS service is usually no longer required. Hence, the DNS filter discards as non-P2P flows all flows that are preceded with a successful DNS resolution.

Failed connection filter processes flows not eliminated by the DNS filter. It observes the churn effect, which is an inherent property of P2P systems and critical to their design and implementation [12]. Churn effect 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 consider as failed connection attempts all unidirectional UDP flows, and failed TCP syn attempts including either no TCP response or a TCP reset. We use this rate within malware and benign traffic in order to discard those that do not implement P2P applications.

Flow size filter keeps only flows that include P2P signaling activity, and discards P2P data flows. The flow size distribution of P2P traffic usually shows discontinuities near small flow sizes, and that characterize P2P signaling activity [24]. It also includes flows with clearly higher flow sizes, usually involving data transfer. Our P2P filter drops all flows whose size exceeds a given threshold that we empirically define based on P2P flow size distributions in [15, 24].

AS-based filtering: P2P botnets constitute overlay architectures that spans multiple autonomous systems (AS). We use the rate of distinct AS numbers within a malware trace in order to discard non-P2P malware. It is defined as the number of remote AS to the total number of flows ratio in a given malware trace. We empirically set a threshold $\tau_{as} = 0.2$ for this rate, based on our malware training set. We discard malware whose rate of distinct AS numbers does not exceed this threshold.

Although these heuristics cannot discard all non-P2P flows, they are reliable enough to characterize the network behavior of P2P applications. They describe invariants in the P2P signaling activity, and so they cannot be easily evaded without modifying the P2P protocol implementation. The output of this filter is a set of P2P signaling flows for each malware or benign P2P application. We use these flows as input to the P2P inspector and classifier modules.

3.2.2. Malware classification

PeerMinor characterizes the network behavior of a P2P application through its signaling activity, which results in different distributions of flows at the network layer. It proceeds first with a flow clustering step that groups together malware P2P flows that implement the same protocol and signaling activity. Then it uses clusters of P2P flows in order to define a P2P footprint for each malware sample. Last of all, it applies unsupervised clustering to malware footprints in the initial learning set in order to build P2P malware families.

We consider as a malware P2P flow both the flow triggered by a malware and its associated peer response. We represent a bidirectional flow using the following features vector: $f_\alpha =< \mathcal{M}_\alpha, proto, B_s, B_r, Pkt_s, Pkt_r, \Delta_t >$. Features of this vector are defined as follows: $\mathcal{M}_\alpha$ is a tag that associates flow f_α with malware $\mathcal{M}_\alpha$; *proto* is a tag that designates the transport layer protocol, being either TCP or UDP; B_s and B_r are the amount of Bytes sent and received within f_α; Pkt_s and Pkt_r are the number of packets sent and received; and Δ_t is the flow duration. We separately build clusters for TCP and UDP flows using the *proto* tag, as these flows clearly carry different signaling activities.

We use incremental K-means clustering in order to build clusters of P2P flows. It increments clusters when the distance of a flow to all existing clusters exceeds a given threshold. We use the Euclidian distance in order to compute the similarity between two separate malware P2P flows, and we set different clustering thresholds for TCP and UDP flows. In fact TCP flows have a higher offset size because of their larger TCP headers and their higher number of packets compared to UDP flows, due to TCP handshake and TCP Acks. Hence, we empirically set TCP and UDP thresholds to 100 and 20 respectively. They

characterize the minimal flow size (400 and 40) to the minimal packets number (4 and 2) ratio for non-empty TCP and UDP flows.

The classifier module builds clusters of flows by comparing P2P flows that were extracted by the filter module from the initial malware training set. P2P flows for a malware sample $\mathcal{M}_\alpha$ may span on multiple clusters. Each cluster contains flows that have similar network features, and so they are likely to carry the same P2P signaling activity and protocol, but that are triggered by different malware samples. We further build a P2P footprint for each malware in our dataset. It specifies the rate of flows for a given malware within each P2P flow cluster provided by our system. In other terms, a malware footprint $\mathcal{F}_\alpha\{c_i\}_{i=1}^m$ is an $m - arry$ vector of size m, where m is the number of P2P flow clusters. Attribute $\mathcal{F}_\alpha\{c_k\}$ for malware $\mathcal{M}_\alpha$ corresponds to the fraction of P2P flows for $\mathcal{M}_\alpha$ within c_k, with respect to the total number of P2P flows in the network trace of $\mathcal{M}_\alpha$. Attributes of a malware footprint are real values in the $[0, 1]$ interval, with $\sum \mathcal{F}_\alpha\{c_i\}_{i=1}^m = 1$.

PeerMinor further uses hierarchical clustering in order to group together malware that has similar P2P footprints so they are likely to use the same P2P botnet topology. It creates a dendrogram where leaf nodes are elementary P2P malware, and the root node is a set of all malware samples. It also uses the Davies-Bouldin index [30] to find the optimal cut in the dendrogram, and thus to obtain the final set of P2P malware families. Each family includes a set of malware samples aggregated within a single node in the dendrogram. Malware families are further used by the detection module in order to associate an infection with a known malware family.

The classifier module builds a *one-class classifier* for each family of malware. It applies the supervised one class SVM learning model [31] to the footprints of malware within each family provided by the classifier module. It characterizes the P2P footprints for malware samples within this family. During detection, PeerMinor collects the network trace for infected terminals detected by the P2P inspector module. It applies P2P filtering and builds a network footprint that it uses to associate an infection with a known malware family. PeerMinor tests this footprint against the one-class classifiers for all malware families in the training set. It associates an infection with a given malware family when its P2P footprint matches the one-class classifier of this family. Yet PeerMinor is unable to classify a bot infection when its P2P footprint matches any, or more than a single malware family. It notifies the security analyst of a new or unknown P2P malware, so it can be submitted to a deeper analysis.

3.2.3. Malware detection

Extracting detection features. The inspection module uses the labeled set of flows provided by the P2P filter in order to build a supervised malware detection system. It applies K-means clustering, using the same features vector f_α, to P2P flows triggered by a given malware or benign P2P application in order to group together flows that carry the same signaling activity. Flows that implement similar functionalities for a given malware or benign terminal are grouped within the same clusters. As opposed to the malware classifier, the inspector module

builds clusters of flows for each individual malware or benign P2P application and uses these clusters as a labeled set to train a supervised learning system.

The output of the k-means clustering process is a multiple set of flow clusters for each malware or benign P2P terminal. PeerMinor further describes each cluster using a comprehensive set of features. Note that 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 [25]. The raw use of those features usually leads to overfitted models that only detect malware in the initial training set. PeerMinor 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. For example, 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)[32]. 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 build a time series that represent the occurrence of flows in each cluster provided by the P2P filter module. It is defined as the ordered list of timestamps associated with all flows in a given P2P flow cluster, and represents the sequence of arrival times for flows within this cluster. We compute the RPDE metric for each time series using the same mathematical model described in [32]. PeerMinor uses this metric in order to assess the degree of periodicity for P2P flows triggered by a given malware or benign P2P application. In addition to the RPDE, PeerMinor also computes 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. Indeed, the mean inter-flow arrival time is a linear metric, as opposed to the RPDE metric that rather applies in the phase space [32].

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 [21]. 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 obtain these distributions as follows. First 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 IP addresses and destination ports, that is the IP addresses and destination ports not appearing in previous sub-intervals. Then we compute the mean and std for these distributions and we add these to our features vector.

The distribution of new IP addresses characterizes the chunk rate of a P2P application. We also compute the distribution of IP addresses, which 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. We add the mean and std of this distribution 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 [15]. 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.

P2P botnet inspection 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, mainly for specific P2P control operations such as keep alive or route discovery, and so they would share similar network footprints. These clusters 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. Hence, PeerMinor implements a supervised pruning process that eliminates malware control operations being shared with benign P2P applications. For each malware in our dataset, PeerMinor keeps only P2P control flows that show distinctive features with respect to other benign P2P applications in our initial learning set.

PeerMinor discards 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 [30] 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 flow 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 flow 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 4.

We tested multiple learning algorithms in order to build our detection model, including SVM, J48 and C4.5 decision tree classifiers [33, 34]. 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.

4. Experimentations

This section describes the design of our experiments and the dataset that we used in order to build and validate our approach. First, we build a P2P botnet detection model, including both inspection and classification, using a learning set of malware and benign P2P applications. Then we evaluate three properties of our system. First we validate our classification system, including precision and recall, against three commercial AV solutions. Then we use a cross-validation method in order to assess the accuracy of our P2P botnet detection model. We also 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.

4.1. Ground truth dataset

4.1.1. Malware training set

We obtained samples of malware traffic from a security company which collects binaries using its own honeypot platform. Our dataset includes one hour of network traffic for malware executed in a dynamic analysis environment. Malware was granted open access to its command and control infrastructure, including updates and command execution. We were provided malware traffic as separate pcap files, labeled each with the corresponding malware md5 hash. In fact we do not have access to malware binaries, but only to their network traces, associated each with the md5 hash for the originating malicious code. The dataset at our disposal includes network traffic for almost twenty thousand distinct malware samples collected during a three months period, between March and June 2012.

Although malware at our disposal belongs to a private dataset, we build a strong ground truth using only malware samples that were correctly detected and characterized by known AV solutions. Therefore, we use the virusTotal API in order to qualify P2P malware in our dataset and to validate the results of our experiments. We searched in virusTotal for md5 hashes in our dataset that match with AV signatures associated with publicly known P2P malware families. In order to obtain a valid ground truth for our experiments, we pick-up network traces only when their md5 labels match with more than 10 known AV signatures for the same P2P malware family in virusTotal. Note that AV scanners usually assign conflicting signatures for the same malware sample. For example, a same Sality malware has a `kaspersky` signature of `Virus.Win32.Sality.aa` and a `trendMicro` signature of `PE_SALITY.BU`. Therefore, we build our ground truth malware classes by matching keywords associated with known P2P malware families, as shown in table 1. We further compare in this section our malware families with signatures provided by three distinct AV scanners. Table 1 summarizes the six distinct P2P malware families that we identified within our malware dataset. Note that after verification in virusTotal, we observed that almost 60% of our dataset consists of Sality (v3 and v4). However, it also includes significant flows for other P2P malware families.

Regarding the classifier module, we aim at experimentally validating three properties using our P2P malware dataset. First, PeerMinor identifies small malware families into a larger set of P2P malware. For instance, it accurately identifies the ZeroAccess family, although it only constitutes 5% of our initial learning set. PeerMinor performs well against imbalanced malware datasets because of its two stage classification model, including a one-class classification stage that processes samples belonging only to the same family of malware. In fact using supervised learning against imbalanced datasets often leads to overfitted models that are biased towards samples occurring more frequently in the initial learning set. Therefore, PeerMinor implements first an unsupervised clustering step using incremental K-means, and which separates malware samples belonging to different families of malware. It then applies supervised learning, using the one-class SVM classifier model, to each family of malware provided by the unsupervised clustering process. Such double-deck classification approach considerably reduces the impact of imbalanced datasets, as shown further in this section. Second, PeerMinor identifies variants of the same malware family that have different implementations of P2P protocol. We validate this property using the example of Sality versions 3 and 4, that were correctly classified by our system. Third, PeerMinor separates families that use the same P2P protocol, but having different P2P signaling activities. It efficiently classifies samples of Zeus v3 and TDSS malware, although they are based on the same Kademlia protocol.

We tested our P2P filter against the initial malware dataset, using the ground truth provided by P2P malware signatures in table 1. The DNS filter reduces up to the third the initial number of malware samples. Indeed, it cannot discard all non-P2P malware because there is multiple other reasons for malware to operate outside the DNS system, including hard coded C&C addresses, scan

attempts and spam. Yet the flow size filter had little impact with only few flows being discarded, and that mostly carry malware spam activities. Indeed the short dynamic analysis time (1 hour) is not long enough for malware to trigger P2P data flows. On the other hand, the P2P filter uses two distinct thresholds, that are associated with the failed connection (τ_{fc}) and AS-based (τ_{as}) filters. We experimentally configured the values for these thresholds using our ground truth malware dataset. We were able to achieve 100% detection accuracy for values of τ_{as} in the interval $[0.1, 0.3]$ and τ_{fc} in the interval $[0.14, 0.6]$. We thus conservatively set these thresholds to the values 0.2 and 0.3 respectively, using our ground truth dataset in table 1. We collected a total number of 535 malware samples that are classified into six distinct P2P malware families, as in table 1.

4.1.2. Benign P2P training set

We obtain benign P2P flows for our training set using traffic that we collected from a well-protected corporate network. It includes netflow packets collected 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 P2P filter in order to discard flows that do not carry P2P signaling activity. We process network flows separately for each single IP address, and we directly build clusters of benign P2P flows that we can use as input to the inspector module. 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 inspection model.

4.2. Malware classifier

4.2.1. Building malware families

We split the P2P malware dataset that we identified using the virusTotal API into two separate groups. The first one includes 85% of our P2P malware learning set, and that we used to build P2P malware families. The second group includes the remaining 15% malware samples, and that we used to test and validate our system. We randomly extracted the malware validation set from each P2P malware family using the ground truth families in table 1. Hence, our validation set included 85 samples extracted from all six P2P malware families. We use the remaining 450 malware samples in order to build our malware classification system. The fifth column in table 1 summarizes the number of samples for each malware family that we use to build our malware classification system.

The P2P classifier builds clusters of flows in order to group together malware flows that implement the same P2P protocol and signaling activity. It applies

incremental k-means to the entire set of malware P2P flows, using the features vector f_α presented in section 3.2.2. The flow clustering module, applied to the 450 malware samples in our dataset, provided a total number of 28 P2P flow clusters, including 22 clusters of UDP flows and 6 clusters of TCP flows. Table 2 illustrates examples of network features for a subset of 8 P2P flow clusters identified by our system.

As in table 2, different signaling activities for the same P2P malware were indeed classified into different clusters. For example, clusters 2 and 3 in table 2 included two separate signaling messages (Peer exchange and Peer announcement) for the same Sality malware. As shown in table 2, Sality has different average request sizes for its two P2P signaling activities (34 vs 20), and so they were classified into different clusters. Clusters 1, 7 and 8 provide yet another example for the Gnutella protocol. We obtained separate clusters for the query and push signaling activities of the same Gnutella P2P protocol. They respectively use UDP and TCP protocols, and they have different network features so they were classified into separate clusters. Note that we may still obtain clusters that implement the same P2P signaling activity and protocol (e.g. clusters 4 and 5 for the same uTorrent protocol). Nonetheless, these clusters show different P2P network features that characterize different implementations of the same P2P protocol by different malware families.

We use the 28 flow clusters in order to build P2P footprints for malware in our dataset. Malware footprints indicate the proportion of P2P flows for a given malware that belong to each of the 28 P2P flow clusters. Table 3 illustrates examples of P2P footprints from two distinct P2P malware, Zeus v3 and ZeroAccess. Table 3 clearly shows how malware of the same family has almost identical P2P footprints and so it would be grouped within the same clusters. For example, malware of the Zeus v3 family has almost all of its P2P signaling flows in cluster 16, while the few remaining flows belong to cluster 1. On the other hand, malware of the ZeroAccess family has almost a third of its P2P signaling flows in cluster 27, and the remaining two thirds in cluster 25. These two malware families would be clearly separated into two clusters by the malware classifier.

The classifier module uses malware footprints in order to build families of P2P malware. It applies hierarchical clustering, using Python, and uses the Davies-Bouldin index to obtain the optimal set of clusters. We obtained a total number of 8 clusters, associated with 8 distinct P2P malware families. We validate P2P malware families using the ground truth classification in table 1.

Six P2P malware clusters were clearly associated with each of the six malware families in table 1. In fact all Zeus v3, ZeroAccess and Kelihos malware samples were classified into separate clusters respectively. We thus consider our clusters to characterize the P2P network footprint of these distinct malware families. On the other hand, samples of Sality malware were split into two separate clusters, including 295 and 37 samples in each cluster respectively. These clusters are likely to include malware that respectively belong to versions v3 and v4 of the Sality family. Yet we couldn't validate this assumption using our ground truth in table 1 because of the conflicting AV signatures for versions of

Family	Samples	Period	Proto	Training	Eval	Flows	P2P flows
Sality v3, v4	386	40 min	Custom	335	51	105178	28071
Kelihos	41	10 min	Custom	34	7	12906	4440
Zeus v3	35	30 min	Kademlia	27	8	8523	4227
TDSS	40	–	Kademlia	30	10	17680	4368
ZAccess v1	33	15 min	Custom	24	9	14328	5676

Table 1: Malware samples by families of malware

Cltr Id	Nb flows	Proto	P2P proto	P2P activity	B_s	B_r	Pkt_s	Pkt_r
1	205	UDP	Gnutella	Query	35	130	1	1
2	937	UDP	Custom (Sality)	Peer exchange	34	610	1	1
3	11944	UDP	Custom (Sality)	Peer announcement	20	600	1	1
4	1674	UDP	uTorrent	find_node	450	970	3	3
5	1427	UDP	uTorrent	find_node	300	630	2	2
6	1164	UDP	Custom (Zeus)	version request	200	645	2	2
7	5778	TCP	Gnutella	push	367	0	4	4
8	504	TCP	Gnutella	push	882	0	8	7

Table 2: Examples of malware P2P flow clusters detected by our malware classifier

Mlwr	Clr 1	Clr 2	Clr 3..15	Clr 16	Clr 17..24	Clr 25	Clr 26	Clr 27	Clr 28
Zeus 1	**0.07**	0	0	**0.93**	0	0	0	0	0
Zeus 2	**0.085**	0	0	**0.914**	0	0	0	0	0
Zeus 3	**0.03**	0	0	**0.97**	0	0	0	0	0
Zeus 4	**0.037**	0	0	**0.962**	0	0	0	0	0
Zeus 5	**0.071**	0	0	**0.928**	0	0	0	0	0
Zeus 7	**0.098**	0.02	0	**0.87**	0	0.01	0	0	0
ZA 1	0.035	0.014	0	0	0	**0.577**	0.04	**0.3**	0
ZA 2	0.078	0.022	0	0	0	**0.592**	0	**0.3**	0.011
ZA 3	0.102	0.011	0	0	0	**0.606**	0.02	**0.27**	0
ZA 4	0.019	0.015	0	0	0	**0.62**	0.06	**0.29**	0.013

Table 3: Examples of P2P footprints for Zeus v3 and ZeroAccess families

Fmly Id	Nbr	Kaspersky	McAfee	TrendMicro
1	295	**win32.Sality: 193** **Win32.Spammy: 29** Unknown: 23	**W32/Sality: 219** Downloader-CPY: 22 Unknown: 4	**PE_SALITY: 223** WORM_KOLAB: 9 Mal_Odra-5: 2 Unknown: 11
2	27	**win32.Zbot: 25** Unknown: 2	**PWS-Zbot**: 27	**Tspy_Zbot**: 27
3	24	**Win32.Sefnit: 17** **Win32.ZAccess: 7**	**Sefnit: 24**	**Troj_Kazy: 13** Troj_Sirefef: 7 Unknown: 4
4	32	**Win32.Kelihos: 27** unknown: 5	**Win32/Kelihos: 23** GenericBackDoor.xf: 8 unknown: 1	**TROJ_FAKEAV: 29** TROJ_INJECTER: 3
5	37	**win32.Sality: 37**	**W32/Sality: 37**	**PE_SALITY: 37**
6	30	**Win32.TDSS:19** Win32.FakeAV: 11	**FakeAlert-JM: 26** **Trojan.Alureon:4**	**BKDR_TDSS: 30**
7	2	win32.Sality win32.killAV	Win32/Nimnul win32/Zbot	PE_fujacks,PE_nimnul
8	2	win32.Sality unknown	unknown: 2	PE_fujacks,PE_down

Table 4: Comparison with `kaspersky`, `McAfee` and `TrendMicro` signatures

	Kaspersky	McAfee	TrendMicro
Precision	83.16%	88.45%	86.5%
Recall	90.8%	89.31%	94.85%

Table 5: Precision and recall against the three AV scanners

the Sality malware. Therefore, in order to refine our ground truth, we checked the update time for AV signatures that were matching each of the malware md5 hashes associated with the Sality malware. We would expect samples for the version v4 of this malware to be more recent *in general* than samples of version v3. We admit that AV update times do not formally validate our classification because we cannot rule out the possibility of newer malware samples implementing version 3 of P2P protocol for this malware. However, signature update times still provide evidence of different version implementations for this same malware family. Yet we observed that 80% of malware in the smaller Sality P2P cluster has newer update times than all other samples in the larger P2P cluster. We believe this is a clear evidence of two families of the Sality malware, that we associate with versions 3 and 4 of this malware family. In fact, versions v3 and v4 of the Sality malware have different implementations of their P2P signaling protocols, and so AV signatures cannot correctly classify these two malware families based on their system behavior. Our system thus offers a complementary approach that classifies P2P malware based on its network-level behavior, which cannot be easily characterized by host-based signatures.

Finally, we obtained two additional clusters, both including two malware samples that belong to different malware families in table 1. These are clearly outliers and so they were misclassified by our system. PeerMinor was indeed able to correctly classify 446 out of 450 malware samples in the initial training set. It clearly outperforms current AV signatures as it achieved near 0.8% misclassification rate.

4.2.2. Comparison with AV signatures

This section validates P2P malware families provided by our system through comparison with signatures from three AV scanners, including `McAfee`, `kaspersky` and `Trend Micro`. Note that we compare in this section only the malware classification capability of our system with other AV scanners. The detection coverage of our system is further discussed in section 4.3. In fact, our system proposes a behavioral approach that classifies P2P malware on the fly while executing in a dynamic analysis environment. We need to verify the cohesion of our P2P malware families using a learning set of known and already qualified malware dataset. For each malware family created by our system, we collect AV signatures for all samples of this family. We compute the precision and recall of our system in order to validate the consistency of our malware classification with respect to all three AV scanners. Experimental results prove the ability of our system to accurately classify P2P malware using only network level information, with no *a-priori* knowledge about the system behavior of malware.

Table 4 compares malware families provided by our system with signatures from three AV scanners. As shown in this table, AV scanners assign different signatures for samples of the same malware families. These signatures usually constitute different aliases for the same malware family. In order to have common evaluation criteria for all three AV scanners, we used the spyware remove website[3] in order to associate all aliases of the same malware families. For example, the signature `win32.spammy` for the first malware family in table 4 is identified by spyware remove as a `kaspersky` alias of `spammer.sality.a`, and so we consider it as part of the sality family.

Table 5 summarizes the classification accuracy and recall against the three AV scanners. Classification accuracy is computed as the average precision rate for all six P2P malware families identified by our system. We introduce the precision rate for a P2P malware family as the ratio of malware samples that have the same predominant AV signature with respect to the total number of samples in this family. The classification recall is computed the same as for the precision rate, excluding samples that are unknown for AV scanners. As in table 5, our system has almost stable precision and recall against all three AV scanners. It enhances by at least 11.5% the malware classification for AV scanners (in case of McAfee which provides the highest precision rate), based on our ground truth in table 1. It also differentiates samples of the same malware family that implement different variants of the same P2P protocol, as for the sality malware which is indeed represented by the same signature by all three AV scanners. Yet it replaces current AV signature aliases with a common behavioral malware classification, as in the example of the third malware family identified by our system in table 4. It provides a common classification for multiple aliases of the same ZeroAccess malware family, including `win32.ZAccess`, `win32.sefnit`, `troj_Kazy` and `troj_Sirefef` aliases.

[3] `http://www.spywareremove.com/`

	Sality v3	**Sality v4**	**ZeroAccess**	**kelihos**	**TDSS**	**Zeus v3**
Accuracy	99.3%	99.1%	94.2%	95%	98%	100%

Table 6: Classification accuracy by malware family

4.2.3. Inline malware classification

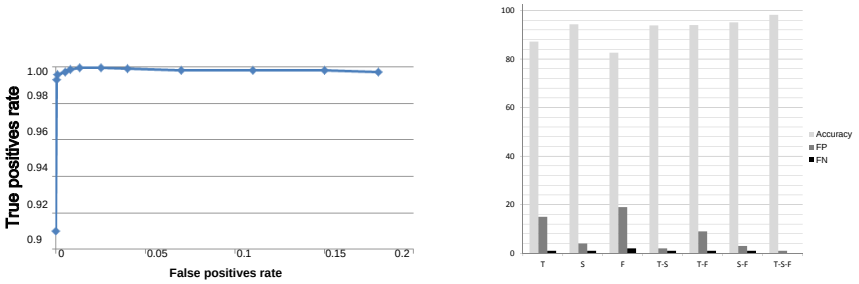
This section demonstrates the classification phase of PeerMinor, which classifies P2P malware on-the-fly using its network footprint. We implement the cross-validation method that consists of extracting an evaluation dataset prior to building malware classifiers, and then to use this dataset in order to test and validate our classifiers. We reiterated the cross-validation process using different evaluation sets, each time randomly extracting 15% of our malware dataset before we build our one-class classifiers. In order to guarantee the soundness of our experiments, our evaluation set had always the same malware composition, as shown in the sixth column in table 1.

We apply the P2P flow filter and we build clusters of P2P flows using the network traces for each sample in our malware validation set. Then we build a P2P footprint for each sample using its P2P flow clusters. We use malware footprints as input to the one-class classifiers for each of our six malware families. Our system achieved near 97.6% classification accuracy, based on the ground truth classification in table 1. The detailed results of our experiments for each malware family are illustrated in table 6.

Samples of Zeus v3 and TDSS malware families were accurately classified with almost no false positives. False positives in case of Sality malware were all due to mis-classifications between the different versions of this family. Note that 100% of Sality malware in our dataset was correctly classified by our system, and almost 99.2% of these samples were classified with the appropriate version of this family. In fact we couldn't formally validate our classification of Sality versions v3 and v4 because AV scanners do not constitute a reliable ground truth. Hence, we used update times for AV detection signatures in order to separate between different versions of Sality malware. On the other hand, PeerMinor has correctly classified only 94.2% of kelihos malware mostly because of the small number of samples in our learning set. Yet PeerMinor outperforms most AV scanners with an overall classification accuracy of 97.6%, while only relying on network features with no need of malware binary analysis.

4.3. Botnet inspection and detection

During the training phase, the inspector module builds clusters of flows for each malware or benign P2P application using flows provided by the P2P filter module. To the purpose of this paper, we processed traffic for the 535 P2P malware samples that constitute our ground truth learning set. We applied incremental k-means, using the same threshold values introduced in section 3.2.2, and we obtained a total number of 1,445 malware P2P flow clusters. On the other hand, we obtained benign P2P flow clusters by applying the same



(a) ROC curve for different pruning thresholds τ_d

(b) Contribution of features towards detection

Figure 2: Evaluation of the P2P botnet inspection model

process to benign P2P flows that we obtained from the well protected corporate network. As described in section 4.1.2, our learning set of benign P2P flows included 794 P2P flow clusters that we used to train our detection model.

4.3.1. Training the inspection model

Our labeled P2P training set included $2,239$ P2P clusters, with up to $1,445$ malware and 794 benign P2P clusters. PeerMinor applies a pruning process that aims at discarding P2P clusters being shared between both malware and benign P2P flows. Hence, it cross-correlated the initial set of $2,239$ P2P clusters, using hierarchical clustering, and obtained 42 meta-clusters. We observed that 35 out of the 42 meta-clusters include up to 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. We believe that these are discriminatory meta-clusters as they clearly characterize malicious and benign P2P activity. On the other hand, 7 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 pruning module as being non-discriminatory flows. We obtained as output to this process a training set of $1,679$ P2P clusters, including $1,192$ malware clusters and 487 benign P2P clusters.

Evaluating detection accuracy. In order to evaluate the detection accuracy of our system, we implemented a cross validation method, using our labeled ground truth dataset. We split our training set into two categories: 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 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 the P2P inspector module, and then we applied our cluster pruning module with different values of the pruning 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 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 figure 2a.

As in figure 2a, 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 PeerMinor. On the other hand, lower values for τ_d allows less discriminatory clusters to go through the pruning process. These would reduce the accuracy of PeerMinor, 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 also used the cross-validation method to evaluate the contribution of our features towards detection. In fact we aim at evaluating the relevance of our features in characterizing and detecting malicious P2P signaling activity on the network. Hence, we built different 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 figure 2b.

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.

4.3.2. P2P botnet detection using 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, including malware classification and inspection, 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 filter was applied at the source, and so we were provided only network flows not preceded with successful 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 identified an overall number of 148 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. Although these signatures do not formally validate our approach, they still provide a ground truth to evaluate our results. P2P signatures detected 169 distinct IP addresses that implement P2P applications, including 21 IP addresses not detected by our filter. In fact 18 of these addresses triggered less than 10 P2P flows. They were discarded by our AS-based filter because of the low AS ratio for these IP addresses. We verified with the ISP the origin of these flows since 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 2 remaining IP addresses detected by the P2P signatures were indeed false negatives. They included utorrent P2P over HTTP flows, and 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 processed P2P flows extracted by our filter using the P2P inspector in order to detect infected P2P bots. Flow clustering implemented by the inspector module provided an overall number of 815 P2P flow clusters. Yet 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[4]. 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. 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 0.4% false positives, associated with two distinct IP addresses during 3 hours of traffic monitoring for $4,347$ distinct IP addresses, which is a fairly reasonable number of alerts to

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

be handled by the system administrator.

The P2P detection module further elaborated a footprint for the infected IP address using the 4 netflow clusters and matched this footprint with the one-class classifiers provided by the classifier module. The resulting footprint was very close to the one for the sality v4 malware and so it was classified by our system as a Sality v4 infection. We manually checked the P2P netflow traces for the given infected IP address. They mainly included UDP flows, where UDP is the main transport protocol for Sality. P2P flows mostly have destination UDP ports above 2200, which is another property of Sality malware according to Symantec [35]. Although the elements at our disposal still provide a convincing evidence, we admit that they do not formally validate an infection by the Sality malware. Unfortunately, we couldn't formally check the infection status on the victim node as the IP addresses were all anonymized and there were no possibility to check on the end node.

5. Discussion

5.1. Malware evasion

Our system detects and classifies malware using statistical features such as flow size, chunk rate, periodicities and botnet distribution. These features are used by both inspector and classifier modules in order to detect and qualify ongoing malware infections. Therefore, we would not be able to accurately detect and characterize P2P bots if the attacker modifies P2P communication intervals, contacts a larger set of peers, or uses random paddings in its malware P2P traffic. Using such techniques could modify statistical consistency in malware P2P flows and so it makes detection more difficult using our features. Although being technically possible, these techniques require a greater effort from malware developers in order to modify their P2P C&C toolkits. They also increase overhead and reduce botnet stability, which makes botnet management more difficult. Yet botnets that use these techniques would no longer be able to dissimulate within benign P2P flows, and so they will be exposed to other malware detection techniques.

5.2. P2P failover mechanisms

Our system only detects and classifies malware that uses P2P protocols as a primary C&C channel. In fact malware may also use P2P protocols as a failover mechanism in case it cannot access its primary C&C channel. Therefore it may not trigger P2P activity during dynamic sandbox analysis, and so it would not contribute to building P2P malware classifiers. In [36], authors propose a dynamic approach that enables to detect secondary C&C channels during malware execution in a sandbox. They propose a mechanism to intercept primary C&C channels and to force malware to engage in a failover strategy. Using techniques such as [36] 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.

5.3. Modularity and elasticity

PeerMinor has a modular and extensible framework that makes easy to add new families of malware to its knowledge database, without altering features for other malware already represented in our system. In fact the classifier module just builds a one-class classifier for the new malware family and adds it to its knowledge base without modifying other existing classifiers. On the other hand, detection features used by the inspector module are built separately for each malware sample. The P2P inspector transparently extracts features for new malware families, without altering other features for malware already known to our system. The P2P inspector further builds a new detection model by applying supervised learning to the new extended set of features. Note that the generation of new detection models is a lightweight process compared to the greedy flow clustering and features extraction mechanism. The latter is not affected while adding a new malware family, and hence detection features are computed only once for every malware family in our database.

5.4. Malware classification

Anti-virus solutions often use proprietary malware descriptors and assign conflicting signatures for samples of the same malware family. Indeed malware classification is becoming a tedious task because of multiple trends in today's cyber threats. Firstly, cyber attackers nowadays collaborate through the black Internet market. They are increasingly sharing source code and using common malware development toolkits. Therefore, malware families oftentimes show similar behavior and use shared exploit kits, which makes difficult to separate between samples of different malware families using only host-based artifacts. On the other hand, modern malware may also download and execute a variety of other malware on the same infected terminal. One example is the sality malware, whose main goal is also to download and execute other malware or attack vectors. Hence, it ends up with multiple malware samples being collocated on the same infected node [35].

Since the gap is being increasingly reduced between samples of different malware families, malware classification using only host-based features is becoming an extremely difficult and error-prone task. Our experiments shown in table 4 clearly illustrate how AV solutions assign conflicting malware signatures. Yet even the same AV scanner may assign different and unrelated signatures for samples of the same malware family. Therefore, PeerMinor outperforms current AV solutions as it provides a malware classification approach that rather observes the network behavior of P2P malware. We believe that malware belonging to the same P2P botnet family implements the same C&C protocols and uses the same encryption suites, otherwise P2P bots would not be able to correctly communicate and share commands on the network. PeerMinor accounts on these observations in order to group together malware samples that belong to the same P2P botnet family. It reliably classifies P2P malware using invariants associated with the way bots communicate and share commands through the overlay network. Our experimental results prove that PeerMinor reliably

classifies samples of the same P2P botnet family using only network features, as opposed to current AV solutions.

6. Conclusion

This paper presented PeerMinor, a fully behavioral system that detects and qualifies P2P malware infections inside a given network perimeter. PeerMinor is a hybrid system that combines both misuse and anomaly-based malware detection techniques. By separating its inspection and classification capabilities, PeerMinor detects both known and unknown malware infections. It associates bot infections with known malware families where possible. Unknown malware infections are identified as such by our system, and so they can be submitted to the administrator for a deeper analysis.

Through its double-deck inspection and classification approach, PeerMinor rapidly discards benign P2P nodes and further classifies malicious P2P nodes in order to provide a comprehensive cartography of malware infections inside a given network perimeter. Experimental results on about 535 distinct P2P malware samples confirm the effectiveness of our system, and show that it can reliably detect and characterize ongoing malware infections.

References

[1] G. Ollmann. Botnet communication topologies: Understanding the intricacies of botnet command-and-control. In *Damballa White Paper*, 2009.

[2] A. Kapoor and R. Mathur. Predicting the future of stealth attacks. In *Virus Bulletin*, 2011.

[3] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is p2p dying or just hiding? In *IEEE GLOBECOM*, volume 3, pages 1532–1538, 2004.

[4] P. O'Kane, S. Sezer, and K. McLaughlin. Obfuscation: The hidden malware. In *IEEE Security & Privacy*, pages 41–47, 2011.

[5] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *Proc. SSP*, 2006.

[6] Trusteer. No silver bullet: 8 ways malware defeats strong security controls. Whitepaper accessible on http://www.trusteer.com/resources/whitepapers, 2012.

[7] C. Rossowz, D. Andriessez, T. Werner, B. Stone-Grossy, D. Plohmannx, C. J. Dietrich, and H. Bos. Sok: P2pwned - modeling and evaluating the resilience of peer-to-peer botnets. In *IEEE Symposium on Security and Privacy (SSP)*, 2013.

[8] J. B. Grizzard, V. Sharma, C. Nunnery, and B. B. Kang. Peer-to-peer botnets: Overview and case study. In *Proceedings of USENIX HotBots*, 2007.

[9] K. Aberer and M. Hauswirth. An overview on peer-to-peer information systems. In *Proceedings of the 4th workshop on Distributed Data and Structures*, 2002.

[10] B. Krishnamurthy and J. Wang. Traffic classification for application specific peering. In *Proc. 2nd SIGCOMM Wrkshp on Internet measurment*, pages 179–180, 2002.

[11] D. Dittrich and S. Dietrich. P2p as botnet command and control: a deeper insight. In *Proceedings of the 3rd International Conference On Malicious and Unwanted Software*, 2008.

[12] D. Stutzbach and R. Rejaie. Understanding churn in peer-to-peer networks. In *Proc. ACM SigComm Internet Measurement Conference*, 2006.

[13] S. Nagaraja, P. Mittal, C.-Y. Hong, M. Caesar, and N. Borisov. Botgrep: Finding p2p bots with structured graph analysis. In *Proceedings of the 19th USENIX Security*, 2010.

[14] C.-C. Wu, K.-T. Chen, Y.-C. Chang, and C.-L. Lei. Detecting peer-to-peer activity by signaling packet counting. In *Proceedings of ACM SIGCOMM*, August 2008.

[15] T. Karagiannis, A. Broido, N. Brownlee, k claffy, and M. Faloutsos. Filesharing in the internet: A characterization of p2p traffic in the backbone. In *UC Riverside technical report*, November 2003.

[16] L. Bilge, E. Kirda, C. Kruegel, and M. Balduzzi. Exposure: Finding malicious domains using passive dns analysis. In *Proceedings of the 18th Network and Distributed System Security Symposium (NDSS)*, 2011.

[17] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel. DISCLOSURE: Detecting Botnet Command and Control Servers Through Large-Scale NetFlow Analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference Network and Distributed System (ACSAC)*, 2012.

[18] M. Antonakakis, R. Perdisci, D. Dagon, W. Lee, and N. Feamster. Building a Dynamic Reputation System for DNS. In *Usenix Security Symposium*, 2010.

[19] J. Francois, S. Wang, R. State, and E. Thomas. Bottrack: Tracking botnets using netflow and pagerank. In *IFIP Networking*, 2011.

[20] G. Gu, R. Perdisci, J. Zhang, and W. Lee. 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.

[21] T.-F. Yen and M. K. Reiter. Are your hosts trading or plotting ? telling p2p file-sharing and bots apart. In *30th Conf. Distributed Computing Systems*, 2010.

[22] J. Zhang, R. Perdisci, W. Lee, U. Sarfraz, and X. Luo. Detecting stealthy p2p botnet using statistical traffic fingerprints. In *Proc. 41st DSN*, 2011.

[23] B. Rahbarinia, R. Perdisci, A. Lanzi, and K. Li. Peerrush: Mining for unwanted p2p traffic. In *10th Conference on Detection of Intrusions and Malware & Vulnerability Assessment*, DIMVA, 2013.

[24] Y. Hu, D.-M. Chiu, and J. C. S. Lui. Profiling and identification of p2p traffic. In *Computer Networks*, volume 53, pages 849–863, 2009.

[25] B. Claise. Cisco systems netflow services export version 9. RFC 3954, Oct. 2004.

[26] N. Kheir and C. Wolley. Botsuer: Suing stealthy p2p bots in network traffic through netflow analysis. In *Proceedings of the 12th International Conference on Cryptology and Network Security (CANS)*, 2013.

[27] N. Kheir and X. Han. Peerviewer: Behavioral tracking and classification of p2p malware. In *Proccedings of the 5th international symposium on Cyberspace Safety and Security (CSS)*, 2013.

[28] Anubis: Analyzing unknown binaries. http://anubis.iseclab.org, 2011.

[29] C. Willems, T. Holz, and F. Freiling. Cwsandbox: Towards automated dynamic binary analysis. In *IEEE Security & Privacy*, 2007.

[30] D. I. Davies and D. W. Bouldin. A cluster seperation measure. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1979.

[31] S. S. Khan and M. G. Madden. A survey of recent trends in one class classification. In *Artificial Intelligence and Cognitive Science*, volume 6206 of *LNCS*, pages 188–197, 2010.

[32] M. A. Little, P. E. McSharry, S. J. Roberts, D. A. Costello, and I. M. Moroz. Exploiting nonlinear recurrence and fractal scaling properties for voice disorder detection. In *Biomedical Engineering Online*, volume 6, 2007.

[33] N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, 2000.

[34] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers, 1993.

[35] N. Falliere. Sality: Story of a peer-to-peer viral network. In *Symantec Security Response Version 1.0*, 2011.

[36] M. Neugschwandtner, P. M. Comparetti, and C. Platzer. Detecting malware's failover c&c strategies with squeeze. In *Proceedings of the 27th Annual Computer Security Applications Conference (ACSAC)*, 2011.