Entelecheia: Detecting P2P Botnets in their Waiting Stage

Huy Hang, Xuetao Wei, Michalis Faloutsos
University of California, Riverside
Email: {hangh, xwei, michalis}@cs.ucr.edu

Tina Eliassi-Rad
Rutgers University
Email: eliassi@cs.rutgers.edu

***Abstract*—Detecting botnets is a critical need for securing one's network and the Internet at large. Despite significant efforts, the problem of botnet detection is still unresolved, especially, when one wants to detect: (a) decentralized or peer-to-peer botnets, (b) botnets that are in a non-active period known as the "Waiting" stage, and (c) polymorphic bots that evade signature detection. We propose a graph-based approach called Entelecheia that is aimed at addressing all three challenges above.**

The inspiration for our work started with the following question: Can we detect botnets by examining long-lived and low-intensity flows? Despite their intuitive appeal, *right out of the box* solutions produce too many false positives. To make it effective, we propose a graph-based solution that focuses on the "social" behavior of the botnet. Specifically, we introduce: (a) the concept of Superflow, to create a graph of likely malicious flows, and (b) two synergistic graph-mining steps to cluster and label botnet nodes. We conduct extensive experiments using real botnet traces injected into real traffic traces. Our approach, *Entelecheia,* produces a median F1 score of 91.8% across various experiments and is robust to various setups and parameter values. Entelecheia can be seen as a first step towards a new and more effective way of detecting botnets.

***Index Terms*—botnet, security, community, graph-mining, anomaly detection.**

I. Introduction

The ability to detect botnets is a crucial component of a network's security system. A *botnet* is a group of compromised computers collective controlled by a *botmaster* that often engages in malicious activities for financial gain such as facilitating spam campaigns or performing Distributed Denial of Service (DDoS) attacks. At the height of its growth in 2007, the spamming *Storm botnet* [9], [20] controlled upwards of twenty million computers and gained enough computational capability (in instructions per second) to rival a supercomputer [22]. In March 2010, Microsoft obtained a restraining order to take down the servers of the Waledac botnet, which infected hundreds of thousands of computers and was capable of sending between 1 and 2 billion spam messages per day [24].

Botnets have two interesting dimensions that are relevant in our work: (a) botnet architecture and (b) botnet lifecycle.

a. Botnet architecture: There are two fundamental approaches in botnet architecture: (a) the centralized approach, which uses Command and Control (C&C) channels such as Internet Relay Chat (IRC) to receive instructions from a single source, (e.g. R-Bot, Spybot, or Gaobot), and (b) the decentralized approach, which utilizes a peer-to-peer protocol to coordinate its operation (e.g. Storm and Nugache). The decentralized (or P2P) approach offers higher resiliency and the botnets implementing it are harder to both detect and take down.

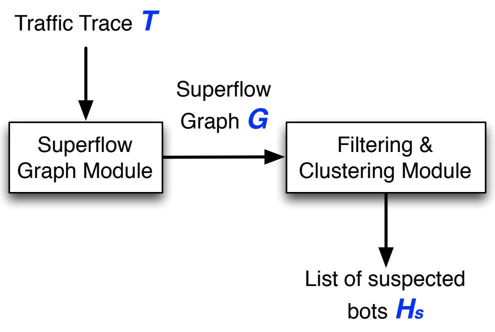


Fig. 1. Entelecheia has 2 modules: *Superflow Graph* and *Filtering & Clustering*.

b. Botnet lifecycle: There are different stages in the life-cycle of a P2P bot [23]: (a) *Infection*, in which the bot spreads, via email attachments, drive-by-downloads, etc; (b) *Rally*, where the bot bootstraps itself by connecting with a peer list; (c) *Waiting*, where the bot waits for the botmaster's command; and (d) *Executing*, in which it actually carries out a command, such as a DoS attack. Each stage has its own distinct behavior, as such customized solutions for each stage can lead to more effective solutions.

We focus on the problem of detecting P2P botnets at the *Waiting* stage with two requirements: (a) we assume no signatures, or equivalently assume that the P2P bot has not been seen before, and (b) we assume no seed information through a blacklist of IPs. The *Waiting* stage is rather "quiet" compared to the other stages, where more aggressive communications takes place. During this particular stage of the cycle, bots are known to maintain infrequent and low-intensity communications with their peers compared to other traffic [25]. It is this behavior that we quantify and exploit in our solution.

To the best of our knowledge, no previous work addresses the aforementioned problem in the challenging context described above. Very few efforts focus on detecting P2P botnets in their *Waiting* stage [7], [5]. The most relevant work is BotMiner [7], which uses signatures and statistical properties to detect botnets to in both Executing and Waiting stages. However, it needs signature analysis and deep-packet inspection, which reduces its effectiveness when signature-matching is not an option, as per one of our problem requirements. Other efforts detect botnets at the more visible *Executing* stage [27], [15], [21], which dovetails with the detection of DoS attacks [10]. Some earlier efforts focus on detecting centralized botnets [3], [17] and modeling the behavior of botnets at different stages [8], [7]. There has been much work on application classification, which is a more general problem. We provide a detailed discussion of previous work in Section VI.

We propose Entelecheia,[1] an approach for detecting peer-to-peer botnets during their *Waiting* stage by exploiting their "social" behavior. The driving insight for our work started with this question: *Can we detect botnets by focusing on long-lived and low-intensity flows?* Despite their intuitive appeal, traditional anomaly detection tools such as [19] introduce many false positives. To overcome this problem, we take advantage of the information hiding in the network-wide interactions of the nodes. Once we create the appropriate graph, we develop graph-mining techniques on it to detect bots. Our approach produces a median F1 score[2] of 91.8%, in a very challenging setting: (a) no signatures available, (b) no initial seeding information, (c) P2P botnets, and (d) during the Waiting stage.

Apart from being a stand-alone solution, we envision our approach as an essential component in a network administrator's toolset against botnets.

Our key contributions are summarized below.

a. A novel graph-based behavior-focused approach. We introduce Entelecheia, an effective approach for detecting bots, the power of which lies in synthesizing: (a) an intuitive observation on how bots operate, (b) the creation of graphs that captures long-lived low-intensity flows, through the definition of Superflows, and (c) two synergistic graph-mining steps to cluster and label botnet nodes. First, we introduce the concept of Superflows, one of the novelties of which is the focus on the interactions between IP addresses only, i.e., using a 2-tuple, instead of the 5-tuple of the common definition of flows. Second, once the Superflow graph is created, we exploit the homophily inherently present in the botnet traffic by identifying clusters of nodes which exhibit high communication persistence.

b. Entelecheia detects botnets effectively. We evaluate Entelecheia on real-world network traces that contained Storm and Nugache [9], [20]. Our approach delivers a median F1 score of 91.8% on a dataset of 20 distinct network graphs, as we explain in Section IV. In fact, the median precision rate that Entelecheia obtained is 98.1%, which shows how accurate Entelecheia is.

c. Entelecheia is robust to variability and statistical "noise". We show that Entelecheia is fairly robust for a wide range of parameter values, natural variability, and statistical noise, all of which are important for a practical deployment.

[1]The name stands for **EN**trap **T**reacherous **ELE**ments through **C**lustering **H**osts **E**xhibiting **I**rregular **A**ctivities.

[2]The F1 score is the harmonic mean of Precision and Recall rates. We will show how to calculate all three metrics in Section IV.

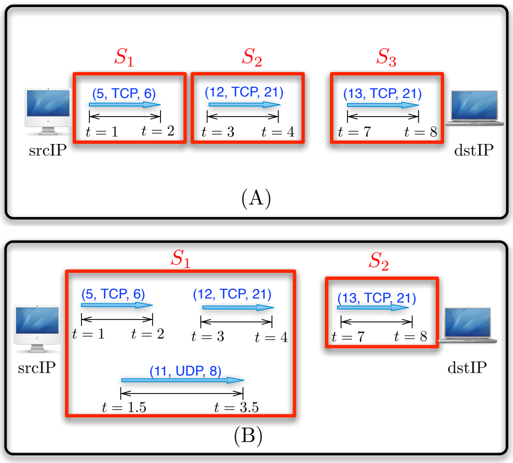


Fig. 2. The 3-tuple above each flow indicates, in order, its source port, protocol, and destination port. **Given an allowable gap of 0.5 seconds between the start and end times of the flows**: (A) The three flows become trivial Superflows. (B) The first three flows can be merged even though their source ports, protocols, and destination ports are not identical.

Table I shows the recommended values and the ranges of the parameters used in our experiments.

We intend to make our code and datasets available to the community in order to contribute towards a systematic and community-wide effort to botnet detection. We have seen first hand that research on botnet detection could benefit greatly from a structured approach that would provide: (a) easy access to real and realistic traces and (b) the establishment of benchmarks for the purpose of evaluating and comparing detection methods.

The remaining of the paper is organized as follows: In Section II, we formally define our novel concept of Superflows. Section III showcases Entelecheia in detail and explains the intuition behind each step that the algorithm makes. In Section IV, we describe our experiments and the effectiveness of Entelecheia. In Section V, we discuss deployment, advantages and limitations of our method. Section VI presents previous work and we conclude the paper in Section VII.

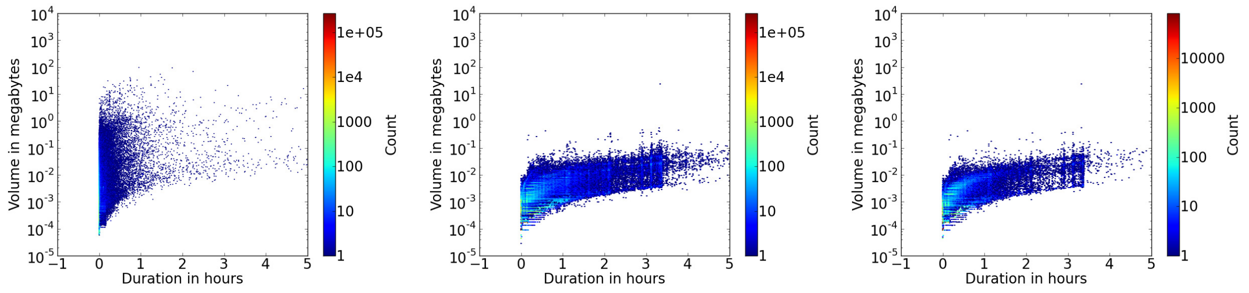
II. Superflows: Definitions and Virtues

We introduce *Superflows,* which helps us hone in on the botnet behavior. Recall that a flow in the traditional sense is defined by a time-stamped 5-tuple: `<srcIP, dstIP, srcPort, dstPort, protocol>`. Intuitively, a Superflow is the maximal group of flows that satisfy the following two criteria: (a) they are between the same pair of nodes irrespective of port numbers or protocols and (b) they are *close in time*. Two flows are *close in time* if their starting and ending times are within T_{gap}, the **interflow gap** parameter.

Figure 2 demonstrates how Superflows are formed in an intuitive manner. There are four steps to how Superflows are constructed for a given trace:

Step 1: We identify flows using the standard definition of a flow.

Step 2: In the beginning, every flow is considered as a trivial Superflow.



(a) 5,000 hosts in WIDE (b) 13 Storm hosts (c) 82 Nugache hosts

Fig. 3. Botnet behavior exhibits long-duration and low-intensity as evident in (b) and (c). These scatterplots show duration versus volume of Superflows in real data traces of backbone (a: WIDE) and botnet-only traffic (b: Storm) and (c: Nugache).

Step 3: We merge two Superflows into a larger Superflow when both of the following clauses are satisfied:

- **Clause 1**: The two Superflows are between the same two nodes regardless of their protocols and port combinations.
- **Clause 2**: The starting time of the second (later) Superflow is within the gapinterval, T_{gap}, from the end of the first (earlier) flow.

Step 4: We repeat Step 3, until there are no more Superflows to merge.

Consider two Superflows, S_1 and S_2, between the same two nodes with starting (s) and ending (e) times: s_1, e_1, s_2, e_2. The duration of S_1 is $d(S_1) = e_1 - s_1$ and the duration of S_2 is defined likewise. For illustration purposes, let us assume that S_1 starts before S_2. The second Clause shown above can be represented by the condition: $s_2 - e_1 \leq T_{gap}$ where $T_{gap} \geq 0$.

Similarly, we define $v(S_1)$ to be the volume (in megabytes) of the Superflow S_1. If S_1 can be merged with S_2, the volume $v(S)$ of the resulting Superflow S is equal to $v(S_1) + v(S_2)$. Note that the starting time of the resulting Superflow S is now s_1 and the ending time is now e_2. It also does not matter if S_2 starts before S_1 ends because in such a case $s_2 - e_1 \leq 0$ and by definition, $0 \leq T_{gap}$.

We next discuss why standard flows are less effective in detecting botnets and present the virtues of Superflows.

a. The commonly-defined flows are less effective in detecting botnets. Previous work [25] showed that using only standard flows does not provide high detection rates (only 87.5% for Storm and 34.8% for Nugache). We will demonstrate in Section IV that using Superflows raises detection accuracy significantly.

Another drawback of standard flows is that in practice flows are not uniquely defined. For example, different flow extractors–such as CAIDA's CoralReef, Wireshark's TShark, CERT's YAF–generate different sets of flows on the same trace because they have different definitions for a standard flow. By grouping flows into Superflows, we remove some of this variability and increase robustness.

b. Superflows capture the behavior of botnet traffic. Figure 3 shows three heat maps. Each dot on a heat map is a Superflow, its X coordinate is its duration in hours and its Y coordinate is its volume in megabytes. Figure 3(a) shows the scatterplot of the Superflows of a randomly selected 5000 hosts in the WIDE traffic traces. Figure 3(b) and 3(c), respectively, show the Superflows of the 13 Storm hosts and the 82 Nugache hosts.

Figure 3 illustrates that regular hosts tend to produce Superflows with short durations and high volumes while malicious hosts generate Superflows that live for hours and have very low volumes. Opportunistic P2P bots communicate with their peers as soon as the former comes online for as long as possible in anticipation of commands from the botmasters. Unsurprisingly, they would periodically send out keep-alive messages to inform their peers that they are online and awaiting instructions.

c. Superflows circumvent a bot's effort to change its communication ports. Upon inspection of the botnet traffic, we saw that unlike the Storm bots, which reused the same source-destination port combination, the Nugache ones changed theirs. So, if we simply grouped flows according to their classic 5-tuple, we would not be able to observe long-lived flows from the Nugache bots because their flows would be put into different groups due to their different source-destination port combinations. Using the full 5-tuple would not affect the detection rate of Storm bots, but it would hinder the effort of identifying the Nugache ones.

Using only the tuple `<srcIP, dstIP>` as the key to flows is a more natural and accurate manner to group the flows together because it bypasses a bot's attempt to change its ports—like Nugache does—and gets to the core of the communication itself: *who is talking to whom?* Furthermore, when a legitimate long-lived, low-volume flow occurs, it is likely to be the result of a *keep-alive* communication between a machine and a server (such as a MSN Live Messenger server). These are continuous TCP flows where the protocol and the source and destination ports do not change. In such cases, source and destination IPs are enough to uniquely identify them.

d. Automating the T_{gap} parameter value selection. The value of the gap, T_{gap}, is crucial for using Superflows in practice. As a rule-of-thumb, the value for T_{gap} should not be so large as to unreasonably consolidate every flow into a Superflow and not so small where flows are not merged into Superflows at all. We automate the selection of T_{gap} by setting it to the median interflow time of the data trace. We chose the median because an inspection of the distribution of Interflow Times showed us that the existence of rare but extremely large values skewed the value of the mean and rendered it unrepresentative of the behavior of Interflow Times.

Specifically, given two flows S_1, S_2 in **Step 4**, we say that the Interflow Time between them is $IT(S_1, S_2) = s_2 - e_1$, where s_2 is the starting time of the second flow and e_1 is the ending time of the first. If S_2 starts before S_1 ends, $IT(S_1, S_2) = 0$.

To select a value for T_{gap}, we collect all Interflow Times between all pairs of successive flows between every pair of communicating hosts and use the **median** value for T_{gap}. This way, T_{gap} is **not** a **free** parameter, as its value adapts to the data trace at hand.

III. Entelecheia: Detecting P2P botnets

Our approach for detecting botnets can be decomposed into two modules that work synergistically to (a) model the network-wide interactions of the hosts in the network as a graph, (b) focus on likely botnet activities, and (c) identify clusters of potentially infected machines. We provide an overview of the modules below.

1. Superflow Graph Module: This module is responsible for creating the Superflow Graph in which each node represents a host in the network and each edge is a vector of attributes that summarizes *all of* the Superflows sent and received between a pair of source and destination IPs. This Superflow Graph represents the *social communication behavior* between hosts.

2. Filtering and Clustering Module: This module first filters out high-volume edges since it is unlikely that they represent Waiting-stage botnet activities. As Figure 3 showed, the volumes of the Superflows from infected hosts tend to be much lower than those from benign ones. By design, P2P bots would form a web of peers among whom the connections are long-lived. If we examine the Superflow Graph and assign as edge weights the total duration of the Superflows (α_d) between the two nodes then clustering nodes together by their connectivity would group infected hosts into communities with a high percentage of long-lived edges.

Below are the detailed descriptions for the aforementioned modules:

A. Superflow Graph Module

The input for this module is a network trace $N = \langle H, F\rangle$ where H is the set of all hosts h_i, for $i = 1, 2, ..., |H|$ in the network and $F = \{f_1, f_2, ..., f_{|F|}\}$ is the set of all conventionally defined 5-tuple *flows*. The output is the Superflow Graph, which represents node-interactions with respect to certain attributes of interest. Algorithm 1 presents the pseudo code for this module. Below, we specify the attributes of interest and offer rationale for their selection.

Algorithm 1 Superflow Graph Module.

```
Require: Trace $N = \langle H, F\rangle$, $T_{gap}$.
 1: $E \leftarrow \{\}$
 2: for all pairs of communicating hosts $h_i$ and $h_j$ do
 3:     Extract $F_{ij}$, the flows between $h_i$ and $h_j$
 4:     Construct $\mathcal{S}_{ij} = \{S_1^{ij}, S_2^{ij}, ..., S_l^{ij}\}$, the set of Superflows originating from $h_i$ to $h_j$, using the $T_{gap}$ parameter.
 5:     $\alpha_v \leftarrow \sum_{k=1}^{l} v(S_k^{ij})$
 6:     $\alpha_d \leftarrow \sum_{k=1}^{l} d(S_k^{ij})$
 7:     edge $e_{ij} \leftarrow < h_i, h_j, \alpha_v, \alpha_d >$
 8:     $E \leftarrow E \cup \{e_{ij}\}$
 9: end for
10: return $G_S = \langle H, E\rangle$
```

1) Identifying Superflows. We parse the given network trace N to aggregate flows into Superflows. This process involves: (a) extracting the set of flows F from the data trace, and (b) merging them into Superflows using T_{gap} as per the definitions given in the previous section.

2) Reporting Superflow properties. At the moment, we focus on two Superflow attributes that are sufficient to enable effective botnet detection: *Volume* and *Duration*. Intuitively, the Volume α_v of an edge between any pair of nodes is the sum of volumes (of the Superflows) between those two nodes throughout the traffic trace. The Duration α_d of an edge between any pair of nodes is the sum of all durations (of the Superflows) between those two nodes throughout the traffic trace. Given that there are n Superflows between hosts A and B, the Volume and Duration attributes of the edge AB are respectively:

$$\alpha_v = \sum_{i=1}^{n} v(S_i), \qquad \alpha_d = \sum_{i=1}^{n} d(S_i)$$

where $v(S_i)$ is the volume of the Superflow S_i and $d(S_i)$ is its duration[3].

In addition, one could expand the number of attributes to more than Volume and Duration. In a more general form, Line 7 of the algorithm can be rewritten as $e_{ij} \leftarrow \langle h_i, h_j, A\rangle$, where A is a vector of attributes of interest, which could include one attribute for the total number of packets, another for packet size variation, etc.

B. Filtering and Clustering Module

Algorithm 2 explains the steps for the Filtering and Clustering module. At a high level, the algorithm operates in three steps:

a. Filtering out edges in the Superflow Graph that are unlikely to be botnet activity. In this step, we want to focus on edges with the profile of P2P-bot behaviors. We only use the volume attribute α_v in this stage, although other attributes of interest can be utilized. In our initial experiments, we have

[3]See Section II

Symbol	Name	Recommended value	Range
T_v	Volume threshold	0.1 MB	0.01MB $\leq T_v \leq$ 0.1MB (Figure 3)
T_d	Duration threshold	1 hour	1 sec $\leq T_d \leq$ 1 hour

TABLE I
THE PARAMETER RANGES IN OUR EXPERIMENTS. SEE TEXT FOR SELECTION PROCEDURE.

Algorithm 2 Filtering and Clustering Module.

```
Require: Superflow Graph $G_S = \langle H, E \rangle$, Volume threshold $T_v$, Duration threshold $T_d$.
    {Select low-volume edges}
 1: $E^{select}$ = {edge $e \in E$: Volume(e) $\leq T_v$ }
 2: $R \leftarrow$ Ratio of edges $e$ in $G_S$ where $d(e) \geq T_d$
    {Cluster the selected graph using Louvain}
 3: $C \leftarrow$ cluster($G(H, E^{select})$, Louvain)
 4: $H_s \leftarrow \{\}$
    {Find suspicious clusters}
 5: for all $c = \langle H_c, E_c \rangle \in C$ do
 6:     $Suspect_c = \{e \in E_c : Duration(e) \geq T_d\}$
        {Calculate percentage of long-lived edges}
 7:     $r \leftarrow \frac{|Suspect_c|}{|E_c|}$
 8:     if $r > R$ then
 9:         $H_s \leftarrow H_s \cup \{nodes\ of\ edges\ in\ Suspect_c\}$
10:     end if
11: end for
12: return $H_s$
```

examined other selection methods such as first selecting on duration and a combination of volume and duration, but those alternatives did not produce any performance gain.

Even when considering only volume, the selection process can be predicated on other attributes. For instance, one could envision different thresholds for TCP and UDP connections. More in-depth discussion on how to select a value for T_d and how its effectiveness varies according to the selection can be found Section IV.

b. Clustering. The choice of which graph-clustering (a.k.a community discovery) algorithm to use is a parameter into our approach, given that many algorithms already exist in the literature [6]. We selected the Louvain algorithm [4] to cluster the Superflow Graph because (a) its notion of a community (with high intra-link density) is well-suited for botnet detection; (b) it is parameter-free; and (c) it is computationally efficient with runtime proportional to $|V| \log |V|$, where V is the set of nodes in the graph. Louvain outputs a set of communities $C = \{c_1, c_2, ..., c_p\}$, where *community* $c_i = \langle H_{c_i}, E_{c_i} \rangle$. H_{c_i} is the set of hosts and E_{c_i} is the edges exclusively within the nodes of cluster c_i. In this paper, we use the words *community* and *cluster* interchangeably.

c. Identifying suspicious clusters and nodes. At this step, the algorithm decides whether a cluster is likely to contain a group of infected hosts by examining the Duration attributes of the edges within the cluster. For each cluster c, we collect all edges e whose $\alpha_d^e \geq T_d$ into the set $Suspect_c$. We next divide the cardinality of $Suspect_c$ by the cardinality of E_c, which is the set of all edges found exclusively inside the community c. This gives us a ratio r that quantifies the presence of long-lived edges in the community.

We say that the cluster c is suspicious if $r > R$. Recall that R is the percentage of long-lived edges as determined by the parameter T_d over the entire Superflow Graph G_S. In our experiments, we varied the value for the threshold R by multiplying it with a ***modulator*** value ranging from 1.5 to 3 to account for statistical variations. While increasing R in this particular fashion improves the precision of our method by a very small amount, the recall rate decreases gracefully as the value of the modulator rises. More details can be found in Section IV.

Careful identification of bots within a cluster. Our initial approach labeled all nodes in the suspicious clusters as malicious. This choice produced many false positives. So, we modified our approach to a two-step process for identifying likely P2P bots in a suspicious cluster. First, we identify suspicious clusters, and within those clusters, we label nodes as malicious only if they transmitted or received long-lived flows.

IV. EXPERIMENTAL EVALUATION

This section is divided into three parts where we introduce the setup of our experiments as well as the datasets we use, the two baseline approaches we evaluated to provide contrast with Entelecheia, and finally the effectiveness of Entelecheia itself. In a nutshell, we observe the following:

(i) Our approach is highly accurate (with a median F1-score of 91.8%) in detecting botnets during their *Waiting* stage.

(ii) Using the 2-tuple is critical for high detection accuracy.

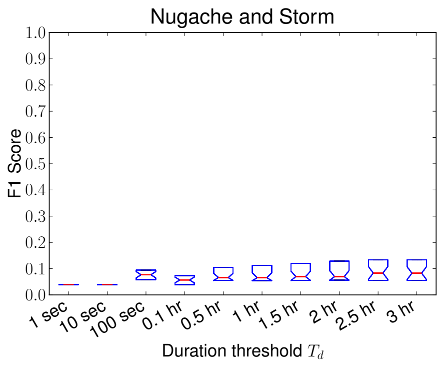
(iii) Our approach is robust for a wide range of parameter values and statistical "noise".

A. Experimental Setup

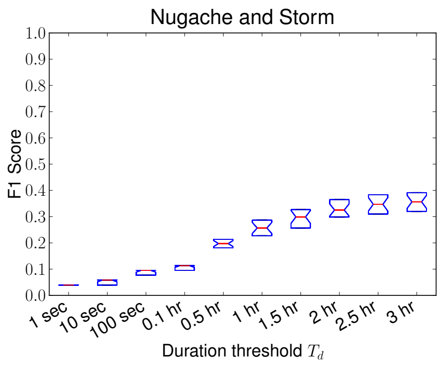
Baseline algorithm: 2-tuple and 5-tuple versions. In our evaluation, we use two reference methods as baselines; and determine whether detecting bots is as easy as simply classifying all long-lived low-intensity connections as botnet traffic.

Our baseline algorithm works as follows. Given a duration threshold and a volume threshold, we label a host as a P2P bot if it produces a Superflow S that is longer than the duration threshold and smaller in volume than the volume threshold. We use two variations of the baseline algorithm, which differ on the communication information used: 2-tuple vs. 5-tuple flows. The **2-tuple** Baseline approach operates on Superflows as defined in Section II. The **5-tuple** Baseline approach operates on 5-tuple flows that are aggregated in the time dimension similar to Superflows (see Section II also).

In this study, we evaluate the following three methods: (a) the 5-tuple Baseline approach, (b) the 2-tuple Baseline approach, (c) and Entelecheia



(a) F1 Score, Baseline, 5-tuple



(b) F1 Score, Baseline, 2-tuple

Fig. 4. F1-score versus T_d (the threshold of what constitutes a long-lived communication) for the two Baseline approaches that use the 5-tuple and 2-tuple definitions. $T_v = 0.1$ MB throughout all experiments.

Parameter selection. Table I lists our parameters (namely, T_v and T_d), their definitions, their recommended values, and the range of values tested for each parameter. To select appropriate values for our parameters, we conducted extensive experiments. We use $T_v = 0.1$ MB for defining volume-intense flows, which proved to be most effective in our experiments. With respect to the duration threshold T_d, we provide results for various values ranging from 1 second to 3 hours. $T_d = 1$ hour balances precision and recall rates, offering very high performance results.

1) Datasets and Analysis Tool: ***WIDE:*** From the MAWI Traffic Archive [2], we downloaded a trace of 24 continuous hours from a Trans-Pacific backbone line between the U.S. and Japan on 03/03/2006 (sample point B). The IPs are anonymized and no packet payloads available. There are 3,528,849 unique IPs and 82,380,945 flows. 89% of all flows are TCP and the rest are UDP. We chose the WIDE dataset because (a) it is freely available online and (b) the length of the dataset spans an entire 24-hour period, which gives us the ease of integration with the Storm and Nugache botnet traces.

Storm and Nugache traces: The real-world malicious network traces that we used in our experiments contain observed data from 13 hosts infected with Storm and 82 hosts with Nugache during a period of 24 hours. The IPs in these traces, which are the same ones used in [7], are not anonymized. In these bot traces, all spamming activities have been blocked to avoid liability issues [25]. As a result, we only see the communication traffic between the bots as they exhibit social behaviors more closely associated with their *Waiting* phase than an active stage of propagation or attack.

Analysis tool: We utilize ARGUS [1] (with its default parameter settings) to process and aggregate packets into flows. We chose this tool because it is widely used and frequently maintained by the developers. Recall that we aggregate flows into Superflows, rendering our approach less vulnerable to the fact that different flow extractors may produce different amounts of flows due to their own definition of a flow and various parameter settings.

2) Generating evaluation traces: We created traces for our experiments by merging the botnet and the backbone traces. At a high level, the process may sound simple; but there are subtleties that must be addressed. In this section, we explain our process and assumptions to make the detection as unbiased and challenging as possible for stress-testing our method.

Overview: We generate 40 different graphs—each comprising of 100-200K nodes—and partition them into 2 datasets: D_1 and D_2, each with 20 graphs. Our goal is to identify 91 malicious nodes[4] among 5000 benign nodes while making sure that we respect the bipartite property of the WIDE trace.

a. Respecting the bipartite nature of the graphs. In overlaying the malicious traces on the WIDE traces, we have to respect their bipartite nature since there are two clear sides in each dataset. A practical deployment of Entelecheia will most likely be on a network of the same property. We envision a detection algorithm to run at a firewall or a backbone link, so we have taken extra care to map all the nodes of each bipartite component to one bipartite component.

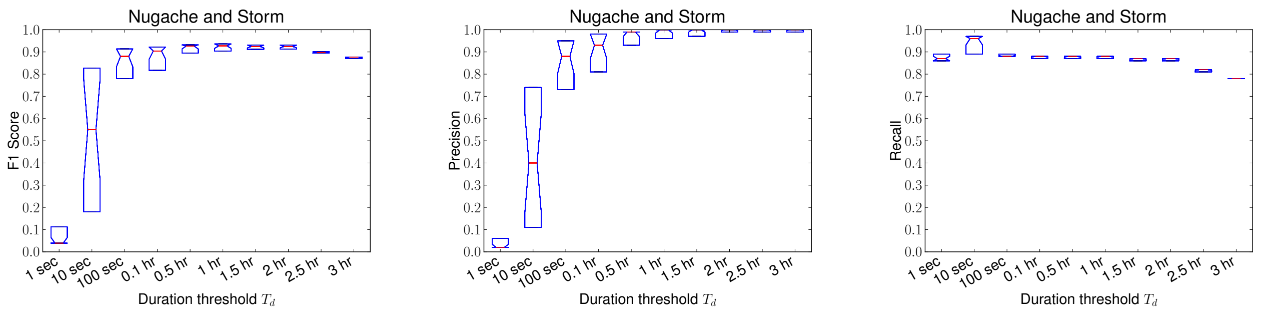
b. Removing the "extra" connectivity from the bot traces. Since the initial botnet traces contain both internal and external edges[5], the graph representing those traces is not bipartite. In our experiments, we remove the internal edges to ensure that the graph formed by overlaying the malicious traces on the WIDE ones remains bipartite. Note that by doing so, we make the detection more difficult for our algorithm, but more similar to an actual deployment of our solution.

c. Sampling the initial trace. Since the backbone trace is large, our first thought was to select a random subset or a time-based interval. Instead, we decided to ensure that the benign graph is connected. The rationale is simple: the more disconnected the graph of the benign nodes is, the easier it is for our algorithm to discover strongly connected communities of malicious nodes. It is for this reason that we need to sample a sufficiently large trace from the initial WIDE traces that would form a graph in which the nodes are connected.

To accomplish this, we start from a random node in the graph representing the full 24-hour WIDE traces and collect nodes via a breadth-first search until we obtained at least a set of 5000 distinct nodes on the same side of the graph to

[4]There is an overlap between the 13 Storm nodes and 82 Nugache nodes.

[5]Internal edges represent the connections among the infected nodes. External edges are the connections that go out to the rest of the Internet.



(a) F1 Score (b) Precision (c) Recall

Fig. 5. Entelecheia's Precision, Recall, and F1 scores.

preserve the trace's bipartite nature. The resulting evaluation trace is comprised of all flows in the 24-hour period of the original trace that involve those 5000 nodes and their neighbors. This typically leads to the formation of a graph with a population of between 100K-200K nodes. Typically, the observed average node degree of a sampled graph is about 3 per node, but there is some variation.

This host selection process is repeated for a total of 40 times (starting with a different random node each time) to produce a set of 40 different traces. Each trace is then overlaid with the botnet traffic in such a way that the known malicious hosts are mapped randomly on the aforementioned 5000 nodes so that the bots are always on the same side of the resulting network graph. Half of those 40 traces (D_1) are used as a training set to help us fine-tune Entelecheia and select appropriate values for its parameters. The other half forms the dataset D_2 that is used to test the algorithm's performance. Each data point (or box) on any boxplots thus represents the median and 95% confidence interval of 20 runs unless otherwise stated. We use **Pr**ecision, **Re**call, and **F1**-scores as our evaluation measures where:

$$\text{Pr} = \frac{\text{TP}}{\text{TP} + \textbf{FP}} \quad \text{Re} = \frac{\text{TP}}{\text{TP} + \textbf{FN}} \quad \text{F1} = 2 \times \frac{\text{Pr} \times \text{Re}}{\text{Pr} + \text{Re}}$$

TP stands for True Positives, FP for False Positives, and FN for False Negatives.

There are two assumptions that we made for the evaluation of Entelecheia:

Assumption 1: For our validation, we only count as True Positives the nodes that we identified as bots who are among the 13 Storm and 82 Nugache bots even though their direct neighbors in the botnet traces are likely to be malicious hosts themselves.

Assumption 2: Given Assumption 1, we further assume that all the nodes in WIDE are benign and report as a False Positive the classification of any such node as a P2P bot.

We revisit Assumption 2 later in Section IV-C.

B. Observations on the Baseline Approaches

Figure 4 shows the performance of the two baseline approaches on D_1. Each data point represents the median of 20 runs and the box indicates the median of the top half (upper quartile) and the median of the lower half (lower quartile). The slanted lines on the boxes indicate the 95% confidence interval around the median.

Observation 1: Using the 2-tuple level of aggregation for Superflows captures botnet behavior more accurately than the 5-tuple. One reason is that using the 5-tuple yields a much lower Recall rate since the Nugache worm changes its source and destination port combinations quite often.

Observation 2: The Baseline approaches perform poorly on D_1 for every value of the duration threshold. The F1-score for the 2-tuple Baseline approach never exceeds 40%. Compared with the excellent performance of Entelecheia (see Figure 5), this result yields two conclusions. First, there exists a large amount of long-lived communications in the WIDE trace. Second, simply flagging long-lived low-volume Superflows is not sufficient to detect infected hosts.

C. Evaluating Entelecheia

Running Entelecheia on D_1. We first evaluate the performance of Entelecheia with various values for the parameter T_d to obtain the best range of values for it. Figure 5 shows the classification performance versus the duration threshold T_d that defines a long-lived Superflow.

We observe the expected tradeoff between Precision and Recall. Precision improves as the time threshold T_d increases, while Recall decreases. This observation further confirms that P2P bots form communities with long-lived, low volume connections. We select $T_d = 1$ hour as the recommended value for T_d since it is in the middle of a plateau where the performance is stable.

Running Entelecheia on D_2. Having picked the value for T_d from our experiment on the dataset D_1, we now run Entelecheia on the second set of graphs, D_2 to evaluate its performance. The results are shown in Figure 6. Note that the median Precision rate is 98.1%, which shows that Entelecheia delivers highly accurate results.

The *True* False Positives: Detecting existing botnet traffic. We investigated Entelecheia's False Positives, namely the source-destination pairs flagged as suspicious, which did not correspond to injected botnet flows. We observed that the flows were directed toward two TCP ports of the destination IPs: 139 and 445. Interestingly, these two ports are associated with security risks on Windows systems. First, threats associated with port 130 include the IRC centralized Spybot, while

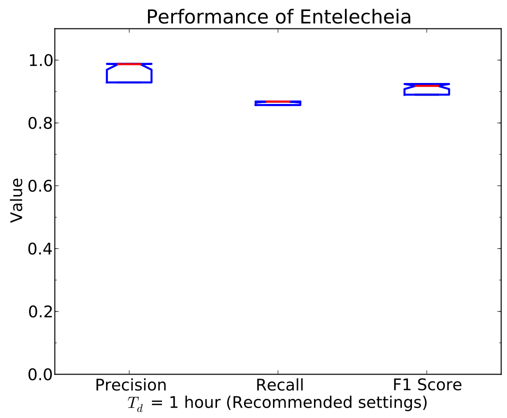


Fig. 6. Performance of Entelecheia on the Test set (D_2): 20 new graphs. Note that the median Precision rate is 98.1%.

worms, such as Sasser, compromised systems via port 445 when it is left open. We are convinced that these FPs are of other previously unknown malicious activities, and argue that the reported accuracy would be higher if we had ground truth for the trace. In the experiment with the **largest** number of False Positives, we have 26 FPs initially, from which 6 (or 23%) are manually verified to be bots or engaging in malicious activities.

The False Negatives. Upon further inspection, we noticed that six of the bots, (6.6% of all bots), that escaped Entelecheia produced very few flows. This is due to two reasons. First, most of their peers were *not* online. Second, the ones that *were* online communicated with the other bots for less than an hour before they went offline. This in turn created short-lived Superflows, which is the reason why Entelecheia could not detect them.

We argue that these infected machines, due to their being disconnected from their peers, are not in the Waiting stage and their having eluded Entelecheia does not undermine Entelecheia's effectiveness because of two reasons: 1) Most of their peers were not online and 2) the ones that were online communicated with the bots for less than an hour before they went offline (the average volume for all of them during the day is 55 bytes). The bots, as a result, produced short-lived Superflows, which is the reason why Entelecheia couldn't detect them. We argue that these infected machines, due to their being disconnected from their peers, are not in the Waiting stage and their having eluded Entelecheia does not undermine Entelecheia's effectiveness.

V. Discussion

Here we discuss several deployment issues, limitations, and ways to improve our approach.

a. Evading Entelecheia: It is interesting to consider how a botmaster could avoid detection by Entelecheia. Such a case would mean a combination of the following three approaches: (a) non-collaborative operation, (b) short-lived connections, or (c) high-volume connections. First, abandoning the non-collaborative operation would be a triumph for Entelecheia, given that decentralized P2P botnet emerged as the answer to the problem posed by the methods designed to expose centralize botnets. Second, changing the last two properties increases the risk of their behavior being caught by an anomaly detection algorithm, which observes changes in the behavior of nodes. In addition, the collaborative operation depends strongly the long-lived connections between peers, therefore it is difficult to change this behavior.

b. Deploying Entelecheia: practical tips and limitations. Our approach has only two and quite intuitive parameters, such as the thresholds for duration and traffic volume. We are confident that the proposed values provide a good starting point and some minor fine-tuning could be beneficial depending on the deployment location and properties of the network. The approach is lightweight as it operates at the packet header level and retains only high level information of the interaction (as opposed to signature based methods that require expensive deep packet inspection capabilities).

In a practical deployment, we envision our approach as a component in a comprehensive security system. For instance, it could serve as the seeding phase for other monitoring tools and approaches. More precisely, Entelecheia can be deployed to identify an initial set of bots, which will be used by another module to extract signatures.

While the anonymized and payload-free WIDE traces prevent us from knowing whether P2P traffic already exists there, we argue that legitimate P2P traffic are high-volume in nature and will be filtered out before the Clustering step. We believe our approach will still work with centralized botnets because their traffic will simply form easy-to-detect clusters with high-duration and low-volume flows.

VI. Related Work

BotMiner [7] is the closest work to our research. It identifies botnets by looking at the bots' activities during different phases of its life cycle. BotMiner has two main components: 1) a C-Plane monitor that converts flows into combined, vectorized records (c-flows) and clusters the records to capture command-and-control traffic and 2) an A-Plane monitor built on the Snort engine[19] that clusters malicious activities (such as spamming and scanning) through Deep-Packet Inspection and signatures. Entelecheia has two advantages over BotMiner. First, Entelecheia does not require signature examination. Second, Entelecheia captures temporal information in the flows, which BotMiner's c-flows do not. Third, Entelecheia does not rely on port information in the flows, which can be easily manipulated by botmasters.

In [16], Li et al. presented their analysis on the prevalence of scanning/probing patterns of bots and presented a statistical approach designed to detect malicious hosts when they are actively probing other hosts in order to discover vulnerabilities. This can be considered a detection method for the *Executing* phase. Entelecheia instead focuses on the *Waiting* one.

In [25], Yen and Reiter introduced the concepts of Traders (genuine P2P users) and Plotters (compromised machines). They presented a way to distinguish the two, which uses the classical 5-tuple definition of a flow. Their approach achieves a

Storm-bot detection rate of 87.5% and Nugache-bot detection rate of 34.8%. We on average achieve a 100% detection rate for Storm and 87% for Nugache.

Zhang et. al. [26] proposes a scheme to separate stealthy P2P *attack* traffic from benign P2P ones by utilizing flow statistics to create statistical fingerprints for flow clusters. They do not use connectivity in their clustering step, presumably because it would complicate the process and makes it harder to separate P2P traffic from other types. Entelecheia utilizes *both* flow statistics and connectivity to deliver accurate detection results.

Nagaraja et al. [18] propose a probabilistic detection scheme that partitions the graph into subgraphs and designate clusters with more homogeneous probability values as malicious. Entelecheia uses the Superflow graph and finds communities based on the standard notion of a community in social networks, which is more analogous to P2P botnets social in nature.

Graph-based approaches have successfully been used in problems where the goal is to identify the applications that generate traffic [13], [14], [11], [12]. Entelecheia focuses on a different problem, namely, detecting botnets in their Waiting stage.

VII. Conclusion

We propose a novel, simple, intuitive, yet effective approach for detecting P2P botnets during their Waiting stage by identifying their "social" behavior. We operate under the following two requirements: (a) we assume no signatures or prior knowledge; and (b) we assume no seed information through a blacklist of IPs. Our key insight is to exploit the inherent behavior of botnets by examining long-lived and low-intensity flows. Identifying what constitutes as a long-lived and low-intensity flow is not a simple task, as we demonstrated through two baseline algorithms.

The algorithmic novelty of Entelecheia is two-folds: (a) the concept of **Superflow** (which filters out flows that are unlikely to be malicious) and (b) a graph-based behavioral approach (which employs clustering techniques). We stress-test our approach using real botnet traces from the Storm and Nugache botnets injected into real traffic traces. Our approach produces a median F1 score of 91.8%. It is robust to parameter values and variations of the setup. This detection performance is noteworthy since our IP network has less than 2% malicious nodes.

Our work can be seen as a first step in developing behavioral-based approach, that can lead to a series of techniques for detecting botnets without signatures and during their more quiet Waiting stage. Moreover, Entelecheia is complementary to other botnet detection approaches such as IP blacklists, deep packet inspection, and detection methods in other stages of their lifecycle.

References

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

[2] Mawi traffic archive. http://mawi.wide.ad.jp/mawi/samplepoint-B/20060303/.

[3] Binkley, J. R., and Singh, S. An algorithm for anomaly-based botnet detection. In *Proc. of USENIX SRUTI* (July 2006).

[4] Blondel, V., Guillaume, J., Lambiotte, R., and Lefebvre, E. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment* (2008).

[5] Coskun, B., Dietrich, S., and Memon, N. Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts. In *Proc. of ACSAC 2010*, ACM, pp. 131–140.

[6] Fortunato, S. Community detection in graphs. *Physics Reports 486*, 3 (2010).

[7] Gu, G., Perdisci, R., Zhang, J., and Lee, W. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. In *Proc. of Usenix Security 2008*, USENIX Association, pp. 139–154.

[8] Gu, G., Porras, P., Yegneswaran, V., Fong, M., and Lee, W. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proc. of 16th USENIX Security Symposium* (2007), USENIX Association, p. 12.

[9] Holz, T., Steiner, M., Dahl, F., Biersack, E., and Freiling, F. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *Proc. of LEET 2008*.

[10] Hussain, A., Heidemann, J., and Papadopoulos, C. A framework for classifying denial of service attacks. In *Proc. of SIGCOMM* (August 2003).

[11] Iliofotou, M., Gallagher, B., Eliassi-Rad, T., G., X., and M., F. Profiling-by-association: A resilient traffic profiling solution for the internet backbone. In *Proc. of ACM CoNEXT* (Dec. 2010).

[12] Iliofotou, M., Kim, H., Faloutsos, M., Mitzenmacher, M., Pappu, P., and Varghese, G. Graption: A graph-based p2p traffic classification framework for the internet backbone. *Computer Networks* (2011).

[13] Iliofotou, M., Pappu, P., Faloutsos, M., Mitzenmacher, M., Singh, S., and Varghese, G. Network Monitoring using Traffic Dispersion Graphs. In *Proc. of IMC* (2007).

[14] Jin, Y., Sharafuddin, E., and Zhang, Z. Unveiling core network-wide communication patterns through application traffic activity graph decomposition. In *Proc. of SIGMETRICS 2009* (2009), ACM, pp. 49–60.

[15] John, J. P., Moshchuk, A., Gribble, S. D., and Krishnamurthy, A. Studying spamming botnets using botlab. In *Proc. of NSDI 2008*.

[16] Li, Z., Goyal, A., Chen, Y., and Paxson, V. Towards situational awareness of large-scale botnet probing events. In *IEEE Transactions on Information Forensics & Security* (March 2011).

[17] Livadas, C., Walsh, R., Lapsley, D., and Strayer, W. T. Using machine learning techniques to identify botnet traffic. In *Proc. of WoNS* (2006).

[18] Nagaraja, S., Mittal, P., Hong, C., Caesar, M., and Borisov, N. Botgrep: Finding p2p bots with structured graph analysis. In *Proc. of the 19th USENIX conference on Security* (2010), USENIX Association.

[19] Roesch, M., et al. Snort: Lightweight intrusion detection for networks. In *Proc. of LISA* (1999), pp. 229–238.

[20] Stover, S., Dittrich, D., Hernandez, J., and Dietrich, S. Analysis of the storm and nugache trojans: P2p is here. *USENIX;login 32*, 6 (2007), 2007–12.

[21] Stringhini, G., Holz, T., Stone-Gross, B., Kruegel, C., and Vigna, G. Botmagnifier: Locating spambots on the internet. In *Proc. of Usenix Security 2011*.

[22] Tung, L. Storm worm: More powerful than blue gene? http://www.zdnet.com/storm-worm-more-powerful-than-blue-gene-3039289226/.

[23] Vacca, J. *Computer and Information Security Handbook*. Morgan Kaufmann, 2009.

[24] Whitney, L. With legal nod, microsoft ambushes waledac botnet. http://news.cnet.com/8301-1009_3-10459558-83.html.

[25] Yen, T., and Reiter, M. Are your hosts trading or plotting? Telling p2p file-sharing and bots apart. In *Proc. of ICDCS 2010*, IEEE, pp. 241–252.

[26] Zhang, J., Perdisci, R., Lee, W., Sarfraz, U., and Luo, X. Detecting stealthy p2p botnets using statistical traffic fingerprints. In *Proc. of DSN* (2011), IEEE.

[27] Zhao, Y., Xie, Y., Yu, F., Ke, Q., Yu, Y., Chen, Y., and Gillum, E. Botgraph: Large scale spamming botnet detection. In *Proc. of NSDI 2009*.