Mining the Network Behavior of Bots

Lorenzo Cavallaro, Christopher Kruegel, and Giovanni Vigna
Department of Computer Science
University of California, Santa Barbara
{sullivan,chris,vigna}@cs.ucsb.edu

July 16, 2009

A botnet is a network of compromised hosts that fulfills the malicious intents of an attacker. Once installed, a bot is typically used to steal sensitive information, send SPAM, perform DDoS attacks, and other illegal activities. Research in botnet detection has been quite prolific in the past years, producing detection mechanisms that focus on specific command and control structures, or on the correlation between the activities of the bots and the communication patterns shared by multiple infected machines.

*We present an approach that aims to detect bot-infected hosts. Our approach (*I*) is independent on the underlying botnet structure, (*II*) is able to detect individually infected hosts, (*III*) deals with encrypted communication, (*IV*) does not rely on the presence of noisy malicious activities and can thus detect legitimate-resembling communication patterns, and (*V*) has a low false positive rate.*

Our technique starts by monitoring a network trace produced by a bot sample $\mathcal{B}$, which is summarized into a set of network flows. Similar flows are then grouped together by relying on a hierarchical clustering algorithm. The resulting clusters are analyzed for evidence of periodic behaviors. If no periodic behaviors are found, an output-based system selects those clusters that recur the most across different network traces obtained by running the sample $\mathcal{B}$ multiple times. Finally, our analysis automatically *produces a network behavior model of $\mathcal{B}$, which is deployed on a Bro NIDS sensor, that operates on real-time and realistic settings, raising few false positives.*

1 Introduction

Malicious software (malware) is code that fulfills the malicious intent of an attacker. The results of a malware infection can be catastrophic. For example, the victim's host may unintentionally participate in coordinated, malicious networks, called botnets. A botnet is a network of compromised hosts, called bots. A bot is a piece of malware that gets typically installed, on a victim host, either by luring the user into landing on malicious web pages that contain drive-by download attacks, or by exploiting simple-but-effective social engineering techniques. Once installed, a bot can be used to steal sensitive information, send SPAM, perform denial of service (DoS) attacks, and other illegal activities.

Bots are different from other malware as they are characterized by the presence of a command and control channel (*C&C*) that is used by the bots to interact with their bot master. Traditionally, *C&C* communications have been carried out following a centralized structure, where push-based protocols, notably IRC, have been used. Soon, bot masters realized that a pull-based approach, which mainly relied on the popular HTTP protocol, achieve a stealthier communication. Eventually, the quest for stealthiness pushed the botnet topology from a centralized structure to a decentralized one, and botnets started to use P2P networks to communicate.

Typically, bot detection, or, more generally, malware detection, relies on anti-virus systems deployed on end-users machines. These systems analyze a malware sample using syntactic signatures that are able to identify a number of different malware instances. Unfortunately, these approaches have several drawbacks. For instance, they are of a limited effect when packing and obfuscation techniques are deployed [4]. These approaches aim to preserve the semantic behavior of the malicious sample, while changing its binary representation. To overcome the limitations of signature-based systems, behavior-based malware analysis and detection techniques were introduced [5,8,20,22,29]. By comparing the actions that a malware performs to those expressed by a model, it is possible to focus the analysis on the matching of semantic features, i.e., behavior, rather than syntactic artifacts. These approaches are very effective in analyzing and detecting malicious samples and describing their behavior. Unfortunately, the time and computational resources required to fulfill such tasks are substantial and, even worse, users are required to install the analysis platform on their machines. Therefore, it would be desirable to have complementary solutions that monitor network events to spot bot-infected machines. In this paper, we present such a solution. Our system monitors the activity of a bot program in a controlled environment and extracts models that capture the characteristic network-level activities of this sample. These models can then be deployed on the network, identifying traffic that is similar to the previously-observed, malicious connections.

2 Related Work and Motivation

Research in botnet analysis and detection has been quite prolific in the past years [3,6,11–15,19,24,26,28].

One line of research devoted to botnet detection revolves around the concept of *vertical correlation.* Basically, network events and traffic are inspected, looking for typical evidence of bot infections (such as scanning) or command and control communications. For instance, BotHunter [14] uses a combination of signature and anomaly-based intrusion detection components to detect a typical bot-infection life-cycle, while Rishi [12] examines IRC-based network traffic for nickname patterns that are frequently used by bots. Alternatively, the approach described in [3] models IRC-based network traffic statistics and checks whether they are suspicious or not. Unfortunately, some of these techniques are tailored to a specific botnet structure [3, 12], or they rely on the presence of a specific bot-infection life-cycle [14]. Moreover, most techniques rely on the presence of noisy behavior such as scan, SPAM, or DoS traffic that has to be observed to trigger an initial alarm.

The other line of botnet detection research focuses mainly on *horizontal correlation*, where network events are correlated to identify cases in which two or more hosts are involved in similar, malicious, communication. In this direction, interesting approaches are represented by BotSniffer [15], BotMiner [13], TAMD [28], and the work proposed in [26]. Except for [26], which detects IRC-based botnets by analyzing aggregated flows, the main strength of the aforementioned systems is that they are independent on the underlying botnet structure, and thus, they have shown to be effective in detecting pull-, push-, and P2P-based botnets. On the other hand, correlating actions performed by different hosts requires that at least two hosts in the monitored network are infected by the same bot. As a consequence, these techniques cannot detect individual bot-infected hosts. This is a significant limitation, especially when considering the trend toward smaller botnets [6]. Moreover, the detection mechanisms of horizontal correlation approaches is usually triggered once malicious noisy behaviors such as scan, SPAM, and DDoS traffic, are observed [13]. This can be a problem, as the past few years have been witnessing a shift of malware from a for-fun activity to a for-profit one [10,16]. As a result, bots are becoming increasingly stealthy. For instance, the connections that Torpig bots open to leak stolen data resemble legitimate HTTP communications that are unlikely to be detected as noisy [25].

The approach we propose in this paper addresses some of the drawbacks of previous bot detection techniques. Our approach is based on the idea that we first observe the network activity of a bot in a controlled environment. Then, based on the recorded traffic, we *mine* the essential network behavior of this bot, i.e., the network communications that a given bot *must* carry out to fulfill its malicious tasks. Examples of this includes *C&C*-like communication patterns and stolen information submissions. On one hand, this provides important insights into the most relevant activities carried out by a monitored sample. On the other hand, this enables us to *automatically* generate network-based behavioral models that accurately characterize and match a bot's network behavior.

Mining the network activity of bots is a technique that our approach shares with others [12–15]. However, our system has a number of salient properties that sets it apart from previous work. First, our system does not rely on any a priori knowledge about the botnet structure. Second, it is resilient to the presence of encrypted communications as it does not inspect packets' content. Third, different from previous research such as [13], our technique detects bot infections even when *only one* individual host is compromised, and, Finally, our technique does not rely on the presence of noisy activities, such as scanning or denial of service traffic. In summary, we make the following contributions:

1. We present a technique to analyze network traffic generated by a monitored bot sample. More precisely, our analysis mines the interesting part of a bot's network behavior, and automatically generates network behavior models that faithfully describe it. Our analysis does not rely on any assumptions on the underlying botnet structure, nor on the communication mechanisms deployed, e.g., plain-text or encrypted communications. In addition, our analysis detects a bot infection even when only one observed machine is infected, i.e., without making any correlation among multiple hosts within the monitored networks. Moreover, our approach detects low-profile malicious behaviors without the need to observe any noisy malicious activity.

2. We develop a prototype implementation of the proposed analysis that is able to accurately generate bot detection models that meet the aforementioned goals.

3. We evaluated our approach on 11 bot samples, including push-based, pull-based, and P2P-based botnets, and 308 additional IRC-based network traces. The generated models detected the bots behaviors with very few false positives, which suggests that our approach can be effectively deployed in real-world networks.

3 Mining the Network Behavior of Bots

Figure 1 depicts the main components of our analysis process, whose goal is to automatically generate a network behavior model of a bot $\mathcal{B}$ (shaded light gray boxes are optional phases). This model describes the interesting network activities that the bot engages in. Such activities characterize core properties of a bot's (network) behavior, without generating too many false positives. For instance, a network behavior model generally reveals *C&Cs* communications and periodic non-noisy activities carried out by bots during their lifetime.

For our analysis, a network trace $\mathcal{T}_{\mathcal{B}}$ of a bot $\mathcal{B}$ is collected. We obtain malicious traces from different sources. For instance, they are collected with the help of honeypots deployed in the wild or via SPAM traps. Once a network trace has been collected, the first step of the analysis process is to summarize the trace in a series of network flows, or connections. At this point, these flows represent the entire network behavior of the observed sample. These are then processed

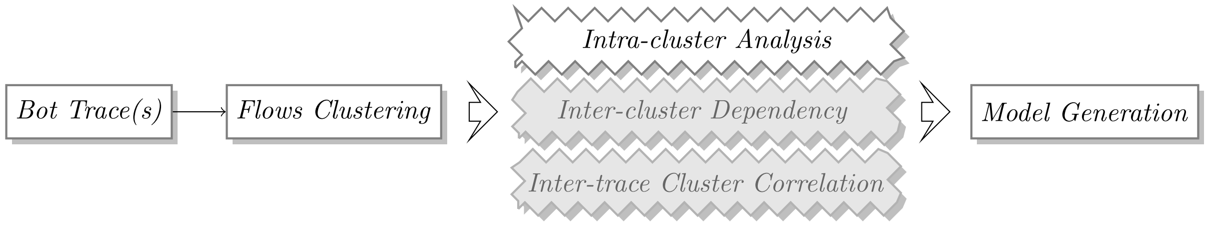

Figure 1: Mining the Network Behavior of Bots: Analysis' Framework.

by a filtering step. The goal of the filtering step is to remove known and noisy attack traffic, such as scan attempts or SPAM. Such attack traffic is typically easy to characterize with certain specifications. For instance, we currently filter out scan attempts by relying on network flows state classification determined by Bro, a powerful, flexible, and event-driven network intrusion detection system [23]. Besides the fact that "uninteresting" traffic is removed from the subsequent analysis process, this step has also the benefit that it speeds up later phases (since less traffic needs to be considered).

The next step is to cluster flows by grouping connections that have similar network features (described in more detail in Section 4). Clustering and grouping related network flows provides an important advantage; it allows us to reason and focus the attention on cluster-level properties as opposed to single flow-level ones. Moreover, it allows us to identify those clusters that are likely to be interesting, i.e., relevant to the behavior of a bot. To find relevant clusters, we leverage information about timing-dependencies among flows within a cluster, action-dependencies among flows of different clusters, or recurring clusters in different network observations of $\mathcal{B}$. This analysis is described in Section 5.

Finally, in the last step, the clusters selected by the analysis are translated into a practical detection model that describes the bot's network behavior. The generated models can be leveraged to detect machines infected by $\mathcal{B}$ with very few false positives (discussed in Section 6).

4 Clustering Network Flows

Clustering is an unsupervised machine-learning technique that groups together similar items, or, in a complementary way, sets dissimilar objects apart [27]. It is a simple yet powerful approach that has been used in many research areas [2,13]. The notion of similarity heavily and intrinsically relies on the features used to characterize the samples and how the distance between samples is defined.

In this paper, we mainly focus our attention on coarse-grained network flow features, instead of fine-grained ones. Coarse features are those that reflect high-level properties of a network flow, without considering details at the packet or even the payload level. For instance, coarse features can be the answers to general questions, such as *"How many bytes have been exchanged by this flow?", "Who is the destination of this connection?"*. On the other hand, fine-grained features describe a more detailed characteristic of a given flow. For instance, such features can be the answers to questions, such as *"How many packets have been exchanged in this flow?", "What are the inter-arrival times of the packets exchanged in this flow?"*. Our flow clustering considers the amount of data transferred per flow, i.e., bytes sent and received, and the number of packets per flow. Different to previous research [13], not much importance is given to the inter-arrival time of packets within a flow and the number of packets per hour per flow. The reason is that coarse-grained features are more difficult to change, and thus, our system is more resistant to evasion.

Table 1 depicts the network features used by our clustering algorithm. The c and f labels in the **Granularity** column tag a feature as either coarse- or fine-grained, respectively.

Network Feature	**Granularity**
Byte Sent	c
Byte Recv	c
# Pkts per Flow	f
Destination IP	c
Destination Port	c

Table 1: Features involved in the flow clustering.

At a first glance, coarse-grained flow features can be considered too general to provide an accurate characterization of a process' network behavior. However, as opposed to recent and related research (e.g., [13–15]), it is worth noticing that our analysis does not observe unknown network traffic where benign and malicious activities are mixed. Instead, our approach focuses its attention on network traces that are known to contain only bot-generated activities. Thus, while it would be reasonable to consider both strategies, in our context, fine-grained flow features may over-specialize the observed behavior. For instance, the inter-arrival times and number of packets per hour can be heavily affected by network congestion, topology, and geographic position between the monitoring environment and the involved hosts. Under this perspective, considering the fine-grained network features of the monitoring environment would most likely produce over-specialized detection models that would not match the inferred behavior when deployed on *different* networks. Nonetheless, some fine-grained network features might be useful. In particular, we included the number of packets exchanged per flow, since they generally depend on the underlying TCP/IP stack implementation. In particular, bots have started to use their own stripped down TCP/IP implementation, making this feature quite effective at characterizing network behavior [9].

For clustering, our current implementation uses a standard hierarchical clustering algorithm with average linkage [27]. Our distance function $d(\cdot,\cdot)$, which captures the similarity between two flows, is a simple weighted Euclidean function over $\mathcal{F}$, which is the features set of all the normalized network flows, as shown in Equation (1). Note that IP addresses IP are not normalized.

$$d(a,b) = \frac{1}{|\mathcal{F}|} \sum_{x,y \in \mathcal{F}} \sqrt{f(x,y)^2} \qquad \forall a,b \in flows$$
$$f(x,y) = \begin{cases} \mid x - y \mid & \text{if } x,y \in [0,1] \\ 0 & \text{if } x,y \in \mathsf{IP} \wedge x = y \\ 1 & \text{otherwise} \end{cases} \tag{1}$$

Clustering network flows is motivated by the observation that the actions of bots are methodically repeated over their lifetime, therefore grouping similar flows together not only has the benefit to group similar (network) behavior together, but also to eventually point out those flows that represent the core behavior of the bot.

The outcome of the clustering process is represented by a dendrogram $\mathcal{D}$. Subsequently, according to the distance function shown in Equation (1) and a given threshold, a cut of $\mathcal{D}$ is produced, yielding $\mathcal{C}$, a set of clusters grouping similar[1] flows. It is possible, and very common, to find clusters that contain a single flow. This is often a result of preliminary actions perpetrated by the bot that are not malicious by themselves. For instance, our experiments showed network connectivity checks, network-time synchronization, Google queries, and one-shot actions performed occasionally during the first stage of an infection. Unfortunately, this may also be the result of evasions perpetrated against bot detection systems. Even if we have not witnessed any of these in our experiments, we direct the reader to Section 7 for a discussion about the resiliency of our approach against evasion attacks.

5 Cluster-based Analysis

To get rid of clusters that are likely not interesting, our analysis only considers $\mathcal{C}_k$, a subset of $\mathcal{C}$ that contains only clusters with at least k network flows in them. In our experiments, k was set to 2. We believe that this is a reasonable and low value that will not discard any interesting parts of a bot's behavior.

Once the analysis has produced $\mathcal{C}_k$, three different analyses are carried out: an *intra-cluster* analysis, an *inter-cluster dependency* analysis, and an *inter-trace cluster* analysis. As depicted in Figure 1, both the inter-cluster dependency analysis and the inter-trace cluster analysis are optional (shaded light gray boxes), and they are invoked whenever the output of the intra-cluster analysis yields no results. The three phases are described in detail in the following sections.

[1]In our experiments, we obtained good results by using a threshold of 0.05.

5.1 Intra-cluster Analysis

Intuitively speaking, periodic communications are usually the result of machine-driven events. A typical example is an email notifier that periodically checks for incoming emails. Of course, the same may apply to a bot that *queries* for its *C&C* host or peers. As remarked in [18], in fact, pull-based bots, e.g., bots with an HTTP-based *C&C* server, clearly manifest *periodic* behaviors. The goal of the intra-cluster analysis is precisely to find such periodic behaviors.

Given a set of events $\mathcal{E}$ of different types, an event instance is defined as (e,t), where $e \in \mathcal{E}$ is an event, and $t \in \mathbb{N}$ is the time-stamp of the event e, i.e., its occurrence in a stream of events. An event $e \in \mathcal{E}$ of a certain type is said to be periodic with period p when e occurs in the observed stream of events every $p \pm \delta$ time units, where δ is a tolerance error [21]. Literature on mining periodic behaviors has been quite prolific in the past years [17,21]. Nonetheless, inferring periodic behaviors can be quite hard, especially when different event types, mixed within each other in a unique stream flow, have to be observed and analyzed. Even worse, noisy activities blended in this flow of data may further hamper the precision of the analysis.

The intra-cluster analysis we describe next aims to infer any periodic behavior within a cluster – and determines what period(s) are being used. To this end, similarly to the way flows are clustered together, the time deltas (differences) between all subsequent events are *clustered*, using an hierarchical clustering algorithm [27]. More precisely, flows in a cluster $c \in \mathcal{C}_k$ are ordered based on their (creation) timestamp. Then, timing differences between the i-th and i+1-th flow are determined, for every flow $i \in c$. These deltas are subsequently clustered, producing a dendrogram $\mathcal{D}_c$. Let $\mathcal{P}_c$ be a cut of $\mathcal{D}_c$ so that deltas similar to each other, according to a similarity function and a given threshold, are grouped together. In other words, $\mathcal{P}_c$ represents a set of clusters that groups together similar timing differences among flows of a cluster c, where $c \in \mathcal{C}_k$ has been obtained by clustering the network flows found in a trace $\mathcal{T}_\mathcal{B}$ of a bot $\mathcal{B}$. The intuition behind this approach is that periodic behavior results in a sequence of events (flows) where many pairs of elements in this sequence have a similar time difference (delta). Thus, the clustering algorithm would identify a large group of deltas that are similar.

If a cluster $c \in \mathcal{C}_k$ contains a *single* periodic behavior, it means that there exists one large cluster $p \in \mathcal{P}_c$ where many (most) deltas are grouped (since they are all similar). More formally, we expect that there is a cluster with a large normalized density of deltas, that is, the number of deltas in a cluster p, normalized with respect to all the deltas found in the correspondent cluster c, is greater than a given threshold d[2]. If such a cluster can be found, we conclude that $\mathcal{P}_c$ is periodic, with a period $\mu \pm \sigma$, where μ is the mean over all the flow deltas in p and σ the correspondent standard deviation.

Of course, it may happen that $\mathcal{P}_c$ contains flows that are

[2]We achieved good results by using the same distance function described in Section 4, and a threshold of 0.15 among flows timestamp deltas.

interleaved with more than one periodic behavior. In other words, it may happen that there are some $p \in \mathcal{P}_c$ that cover many deltas. Even if we did not encounter such a situation in the experiments we carried out, our analysis is general enough to account for it. More precisely, clusters in $\mathcal{P}_c$ are ordered based on their normalized density, producing an ordered set of clusters $\mathcal{P}_c^o$. The idea is to consider any cluster $p \in \mathcal{P}_c^o$ whose density is greater than a threshold k, until the cumulative density of the chosen clusters passes the aforementioned threshold d. In other words, our analysis looks for those clusters p that satisfy the following equation:

$$\sum_{p \in \mathcal{P}_c^o \wedge |p| > k} |\, p \,| > d \qquad \forall p \in \mathcal{P}_c^o \tag{2}$$

If (2) holds, then we compute the parameters μ and σ that characterize the periods of the corresponding cluster p. Of course, a sensitivity analysis of the parameters k and d would be needed to pick the values that most likely stabilize the expected result (e.g., low false positive and negative). We have not performed such analysis as we have only observed single-periodic behavior associated with the flows in a particular cluster $c \in \mathcal{C}_k$.

The output of the intra-cluster analysis is $\mathcal{C}_{periodic} \subseteq \mathcal{C}_k$. This set contains only those clusters that show periodic behaviors. Of course, it may happen that no well-defined periodic behavior or timing-dependencies exist for a bot. In such cases, our analysis attempts to determine whether the timing features of the flows in a cluster are drawn from an unknown distribution. Broadly speaking, this result is obtained by performing a statistical test on the timing features of clusters belonging to different network traces generated by a given bot sample $\mathcal{B}$. This inter-trace correlation analysis is discussed in more detail in Section 5.3.

As previously mentioned, it is worth noting that periodic events are very common in pull-based (e.g., HTTP-based), bot communications. At a first glance, it seems unclear whether this periodic behavior is to be found in push-based (e.g., IRC-based), botnets. Such communications, in fact, generally involve a single, persistent connection that connects a bot with its command and control server. Interestingly, however, the periodic behavior can be found by observing flows' idle-times. For instance, periodic PING/PONG-like messages are quite frequent in IRC-based *C&C* communications. Therefore, the intra-cluster analysis technique described above can be easily applied after timing differences between idle-periods of a flow are determined, for all the flows in a cluster $c \in \mathcal{C}_k$. A flow enters the idle state if it has been silent for at least δ consecutive time unit (e.g., seconds). A flow exits the idle state as soon as it starts exchanging data. A flow idle-period is defined as the difference between the time when the flow exited and entered its idle state. Setting δ to 10 seconds yielded good results., as supported by evaluating our approach on 308 traces of IRC-based bot families.

5.2 Inter-cluster Dependency Analysis

In addition to periodic events, bot traces can also contain other low-noise, non-periodic activities. When possible, such activities should also be taken into account. In other words, it would be interesting to "revive" any flow that was discarded because of its a-periodicity, or due to its membership to a cluster deemed to be uninteresting early, i.e., $c \notin \mathcal{C}_k$.

The inter-cluster dependency analysis aims at inferring such properties in a very simple-yet-effective way. To this end, this phase produces $\mathcal{C}_{reconsidered}$, a set of clusters whose flows are found to be dependent on some of the flows present in any of the clusters in $\mathcal{C}_{periodic}$. In other words, a cluster $c \in \mathcal{C}$ is selected if it is possible to find a flow $f \in c$ whose destination (IP address) matches the one of a flow $f' \in c'$, where $c' \in \mathcal{C}_{periodic}$. For instance, in our experiments, this analysis identified cases in which a bot downloaded additional malicious components from a host that was previously identified as a command and control server.

5.3 Inter-trace Cluster Correlation

The intra-cluster timing analysis provides interesting information on actions periodically performed by a bot. The inter-cluster dependency analysis, instead, can identify additional clusters that are related to periodic activities. Unfortunately, as previously mentioned, bots may not manifest well-defined periodic activities at all. Even worse, timing relationships may just be too hard to be accurately inferred. In addition, bots may manifest occasional malicious behaviors that, although interesting, rarely happen. Of course, these problems should not affect the robustness of a behavioral detection model. A robust analysis needs to address these issues and outputs the clusters that can best characterize the bots' malicious behavior.

To acquire a good confidence on the *goodness* of the clusters produced by the previous analyses, we perform an inter-trace cluster correlation analysis. To this end, a bot $\mathcal{B}$ is run different times, producing distinct network traces $\mathcal{T}_i$, for each execution run i. Intuitively speaking, the inter-trace cluster correlation analysis aims at finding similarities among the clusters created during the different runs of $\mathcal{B}$. The more frequently a cluster is observed across different executions, the more it is likely to describe important parts of the bot's behavior. This inter-trace correlation has several benefits. Similar to any output-voting system, it increases the confidence about the quality of an answer, which, in our context, is given by a cluster c. Furthermore, it suggests a way to improve the accuracy of a cluster by correlating some of its features with those of similar clusters obtained during different executions of $\mathcal{B}$.

After $\mathcal{T}_i$ network traces are obtained, the earlier steps of our approach take place, as described in the previous sections. Hence, the traces $\mathcal{T}_i$ are normalized, pre-filtered, and the initial hierarchical clustering process is started, eventually producing the set of clusters $\mathcal{C}^i$, for each network trace $\mathcal{T}_i$. Subsequently, the intra-cluster and inter-cluster dependency analyses are performed, eventually producing the set of clusters $\mathcal{C}_k^i$, $\mathcal{C}_{periodic}^i$, and $\mathcal{C}_{reconsidered}^i$, accordingly to the terminology used so far.

It is at this point that the inter-trace cluster correlation analysis begins. As a general idea, a cluster $c \in \mathcal{C}_k^i$ is finally selected as a strong representative of $\mathcal{B}$'s behavior if clusters *similar* to c exist for the majority of the clusters set $\mathcal{C}_k^i$. Or, more formally, $c \in \mathcal{C}_k^i$ is selected if, $\exists c' \in \mathcal{C}_k^j, i \neq j \mid c' \sim c$, and this holds at least $\frac{i}{2}+1$ times.

A cluster c' is said to be similar to a cluster c, i.e., $c \sim c'$, if c and c' have similar network features and similar timing relationships. To this end, every cluster $c \in \mathcal{C}_k^i$ is *compressed*, producing a cluster c_c, where the features of its network flows are represented by their average, μ_c, and standard deviation, σ_c. Then, a cluster $c'_c \in \mathcal{C}_k^i$ is considered *network-similar* to a cluster c_c if $\mu_{c'_c} \in [\mu_{c_c} - \sigma_{c_c}, \mu_{c_c} + \sigma_{c_c}]$. Clusters with no timing relationships can thus be selected at this stage.

Of course, timing relationships, if any, have to be considered as well before concluding that any two clusters look alike. To this end, the following cases are distinguished:

(a) $c \in \mathcal{C}_p^i$ and a well-defined time-related period(s) exist. We recall from Section 5.1 that a cluster c is periodic if the Equation (2) is satisfied. Then, c is selected and confirmed as a periodic cluster, if

$$\exists c' \in \mathcal{C}_p^j, i \neq j \mid c' \sim c \wedge c' \text{ satisfies Eq. (2)}, \quad \forall \mathcal{C}_p^i$$

and this holds at least $\frac{i}{2}+1$ times. If these relationships are satisfied, then c is added to $\mathcal{C}_{periodic}$.

(b) $c \in \mathcal{C}_k^i \setminus \mathcal{C}_{periodic}^i$. Since $c \notin \mathcal{C}_{periodic}^i$, it means c does not manifest well-defined period(s) and the Equation (2) is not satisfied. Nonetheless, as anticipated in Section 5.1, it is desirable to mine whether the flows in c happen to follow an unknown probability distribution. In fact, the more features one or more flows need to match, i.e., network and timing relationships, the more accurate the behavioral detection model is, and the less false positives our analysis will manifest. To this end, the intra-cluster analysis described in Section 5.1 collects Δ_c, the flows timestamp deltas of $c, c \in \mathcal{C}_k^i \setminus \mathcal{C}_{periodic}^i, \forall c, \forall i$. Let $\mathcal{C}_{dist}^i = \mathcal{C}_k^i \setminus \mathcal{C}_{periodic}^i$. Then, c is selected and confirmed as a cluster whose flow creation follows an unknown probability distribution, if

$$\exists c' \in \mathcal{C}_{dist}^j, i \neq j \mid c' \sim c \wedge \text{K-S}(\Delta_c, \Delta_{c'}) \text{ holds}, \quad \forall \mathcal{C}_{dist}^i$$

and this holds at least $\frac{i}{2}+1$ times, where K-S$(\cdot,\cdot)$ is the Kolmogorov-Smirnov test [7]. Similarly to the previous point, if these relationships are satisfied, c is then added to $\mathcal{C}_{periodic}$.

(c) If the previous points are not satisfied, then c is added to $\mathcal{C}_{interesting}$, a set of interesting clusters, only if the output-voting system driven by the inter-trace cluster correlation analysis yields a positive result (i.e., similar clusters are selected based on network features match only if this answer is obtained in the majority of the cases). More formally, c is added to $\mathcal{C}_{interesting}$ if

$$\exists c' \in \mathcal{C}_{dist}^i \setminus \mathcal{C}_{periodic}^i \mid c' \sim c$$

and, similarly to the previous cases, this holds at least $\frac{i}{2}+1$ times. A similar situation is carried out for clusters belonging to $\mathcal{C}_{reconsidered}$. All the remaining clusters are discarded.

The inter-trace cluster correlation analysis outputs $\mathcal{C}_{periodic}$, $\mathcal{C}_{reconsidered}$, and $\mathcal{C}_{interesting}$, representing the sets of clusters that exhibit timing relationships, and, under a different perspective, interesting behaviors.

Once the inter-trace cluster correlation is finished and clusters are chosen to either be mandatory or optional, a further decision can be made, where similar clusters are merged. This decision could not be taken without relying on the inter-trace cluster correlation for the following reason. Let c_A be a cluster with flows directed to a host A. Let x_A be the feature set of c_A. Similarly, let c_B be a similar cluster with the same feature set but with flows directed to a host B. While it would be reasonable to conclude that c_A and c_B are semantically equivalent, i.e., they refer to different hosts offering the same service, the opposite may be true as well. If the latter holds, the intra-cluster timing-analysis may become polluted if c_A and c_B are grouped together. For instance, c_A may exhibit a periodic behavior while c_B may not. If $\mid c_B \mid \gg \mid c_A \mid$ then the timing analysis can be corrupted. For this reason, the feature set includes the destination IP as a network feature. On the other hand, c_A and c_B may indeed be semantically equivalent. Thus, we merge clusters where flows have similar features if either one of the following holds:

- there are not timing relationship (no periodic, no positive K-S test after the inter-trace cluster correlation) between the considered clusters with similar features values, or
- there are timing relationships and those are similar in the clusters with similar features values.

As we will see in Section 8, the first condition turned out to be useful for accurately modeling the network behavior of the Storm botnet [18], while the second one was useful for modeling the network behavior of the Torpig [9].

The merging strategy also implicitly stands against simple evasion techniques where a bot sends similar (from a feature values perspective) semantically *non-equivalent* flows with the intent to force the analysis to cluster them together and potentially polluting any timing relationship manifested by the bot itself.

6 Model Generation

The cluster-based analysis outputs the clusters set $\mathcal{C}_{periodic}$, $\mathcal{C}_{reconsidered}$, and $\mathcal{C}_{interesting}$, as described early. Let $\mathcal{C}$ be $\bigcup_{j \in \{periodic, reconsidered, interesting\}} \mathcal{C}_j$. Clusters $c \in \mathcal{C}$ are then *compressed*. A compressed cluster c is a cluster with one flow, r_c, whose network features are represented by the average, μ, and the variance, σ^2, over all the flows grouped by c (different destination IPs and ports are, of course, omitted). In other words, r_c represents all the flows in c. It is worth noting that timing relationships, if any, are a cluster-based property and thus they still belong to c, even after it has been compressed.

After clusters obtained by analyzing the network traces of a bot $\mathcal{B}$ are being compressed, it is possible to generate a $\mathcal{B}$'s behavioral detection model. This phase outputs the information contained in the clusters set along with the code responsible for the detection logic. The model is specified as a Bro *policy script* [23] whose details are provided in Section 6.2. In the following, instead, we detail how a behavior model is matched when monitoring unknown network flows.

6.1 Bot Behavior Detection

Intuitively speaking, the detection of a malicious behavior expressed by the corresponding cluster-based model deals with *flows membership*, and *timing-properties verification* We detail such phases in the following.

6.1.1 Flow Membership

During this step, a decision is made to check whether an observed flow f can be assigned to one or more clusters in $\mathcal{C}$. In particular, let c be a compressed cluster of $\mathcal{C}$. Let also r_c be the representative flow of c with network features parameters μ_r and σ_r^2. Then, a flow f matches r_c's network features if the network features of f are within $\mu_r \pm k\sigma_r$. Then, by the Tchebycheff inequality [7], we know that

$$P(\mid x - \mu_r \mid > k) < \frac{\sigma^2}{k^2},$$

that states that the probability that the value of a network feature x is distant from its average μ_r for more than a threshold k is bound by its variance, σ_r^2, and the threshold k. If the initial hierarchical clustering of the flows in the malicious network trace is of good quality, then, by construction, σ_r^2 is small. Consequently, if k is chosen to be equal to the threshold used for clustering network flows, as explained in Section 4, there is an high likelihood the network features of malicious flows will fall within the range $\mu_r \pm k\sigma_r$.

6.1.2 Timing Properties Verification

If one or more flows f are assigned to one or more compressed clusters $c \in \mathcal{C}_{periodic}$, their timing properties need to be verified. Depending on the timing properties of c, different actions may be carried out.

- $c \in \mathcal{C}_{periodic}$ and a well-defined time-related period(s) *does* exist. The period p of c is described by the relations expressed by the equation (2). In particular, the parameters μ_p and σ_p^2 represent, respectively, the average and the variance of p, determined during the intra-cluster analysis described in Section 5.1.

  The difference between the creation time-stamp of two consecutive flows assigned to c is computed. Then, the resulting delta δ is compared to the parameters μ_p and σ_p^2. If $\delta \in [\mu_p - k\sigma_p, \mu_p + \sigma_p]$, then the timing-relationship of c are verified.

- $c \in \mathcal{C}_{periodic}$ and a well-defined time-related period *does not* exist. In this case, a certain quantity k of flows needs to be assigned to c, before verifying their timing properties. To this end, we set k to be equal to a fraction of the number of flows assigned to c during the inter-trace cluster correlation analysis. Then, let Δ_m be a sequence of the flows creation time-stamp deltas determined during the cluster-based analysis. Likewise, let Δ_k be a sequence of the creation time-stamp deltas of k flows observed during detection. If the Kolmogorov-Smirnov on Δ_m, and Δ_k is not satisfied, then Δ_k and Δ_m are most likely drawn from a different probability distribution and c's behavior is not matched.

During detection, we say that a flow f that belongs to a host $\mathcal{H}$ matches a cluster c of a bot's profile $\mathcal{P}$ if f can be assigned to c, and if, by considering f, the timing properties of c, if any, can be verified as well (of course, in the simpler scenario, at least two flows that belong to $\mathcal{H}$ must be assigned to c). However, having a match *does not* trigger an alarm by itself, as we tolerate fewer spurious matches that may happen. Therefore, a bot behavior $\mathcal{P}$ is matched by a possibly infected host $\mathcal{H}$, and an alarm is raised, whenever the following holds:

1. There exists a flow f of $\mathcal{H}$ that matches $c, \forall c \in \mathcal{P}$, and

2. A cluster c of $\mathcal{P}$ has a number of matches that is either proportional to the number of flows clustered during the initial clustering of network flows, or that is proportional to the number of deltas determined during the intra-cluster analysis, $\forall c \in \mathcal{P}$. To make things simpler, our current requirement is that a cluster c must have been matched by flows of $\mathcal{H}$ at least 3 times.

In the following, we provide some details about how a bot behavior is expressed in a practical detection model.

6.2 Mapping Models to NIDS events

Network-based behavior models need to be translated in NIDS events to allow for malicious behavior detection. To this end, we decided to use Bro, a Network Intrusion Detection System (NIDS) which offer a powerful event-driven policy language for writing NIDS policy scripts able to match complex network behavior [23].

This turned out to be an interesting choice as clusters can be easily represented by sets and tables, and custom timers as well as Bro's events, such as `connection_termination`, `connection_partial_finish`, `udp_request`, and `udp_reply`, can easily express the network behavior models generated by our cluster-based analysis.

In fact, a typical network behavior model is composed by clusters and by decisions that have to be taken upon the verification of a particular event (e.g., connection termination). Thus, a typical Bro-based model that implements one of our behavior model uses, (a) records, tables and sets to express clusters along with their network and timing features, and (b) network events that trigger the matching process.

7 Discussion

One of the main issue that our analysis shares with other dynamic-based behavior monitoring approaches is whether the observed behavior is meaningful enough to account for the majority of the infections for a particular bot. From a practical perspective, bots leak information only when such information is available [25]. Otherwise, they are often engaged in ping-like message exchanges with the intent to communicate to a *C&C* or their peers that they are alive.

Triggering meaningful and specific behaviors is a known open and hard research area. One may wonder if the behavioral network detection model our approach generates contains those traits that are most likely shown by a bot. Unfortunately, it is not straightforward to provide a definitive answer to this question. Nonetheless, we think it is reasonable to believe this generally happens. For instance, the majority of the submissions that the Torpig bots recently made to their *C&C* during a ten-days of monitoring were just ping-like submissions reporting no stolen data. Talking about numbers, this translates to $32,980,083$ ping-like submissions over a total of $34,042,098$ [25].

Our analysis tries to characterize a bot malicious behavior by observing the actions it perpetrates. As every anomaly-based system, our analysis can be subjected to evasion attacks. In particular, an attacker might *obfuscate* the communication parameters in order to hamper the clustering approach, which is the basis of the rest of our analysis.

Our cluster-based analyses heavily rely on the quality of the initial hierarchical clustering approach. If flows are not aggregated in meaningful way, then further analyses are ineffective. Flows are aggregated in a meaningful way when the network features used to determine flows similarity are not subjected to any form of obfuscation, or the obfuscation they are subjected to is below the threshold used by the distance function of the clustering algorithm [27]. An active attacker may randomize these features with the intent to confuse the clustering approach and aggregate dissimilar flows together. Of course, the contrary holds and randomization could be used to the extent under which no flows are grouped and every flow forms a cluster on its own. This attack is a limitation of the effectiveness of our approach that, however, shares with similar traffic and network-based detection techniques as well [13, 26, 28]. A possible solution would be to characterize such an attack by adapting the Kolmogorov-Smirnov test we used for capturing intra-cluster timing relationships. Moreover, the same obfuscation evasion could be modeled by an entropy-based analysis which will characterize the entropy level of the network features of a monitored bot during behavioral model construction.

8 Evaluation

In the following, we evaluate the network behavior-based detection models generated by our cluster-based analyses. The main goal of this evaluation is to show that our detection models are able to detect bots with a very low false positive rate. In particular, our models detect malicious behavior even when a single host is infected in the monitored network. In addition, malicious behavior is detected even when only encrypted communications are involved. Finally, our models detect low-pace and non-noisy malicious activities as well.

We first retrieved bot samples from Anubis [1], which receives malicious samples from different sources (e.g., user and honeypots submissions as well as contribution from malware analysis organizations). These samples belong to different botnet structures, e.g., push- and pull-based, as well as P2P. In particular, we collected HTTP-based, IRC-based, and P2P-based bots. Of these, two HTTP-based bots were variants of a unique bot, i.e., Torpig [9], while all the P2P traces were Storm variants [18]. These samples were then executed in a monitored environment for several days.

Table 2 provides details on the behavioral models generated by our cluster-based analyses. It is worth noting that the inter-trace cluster correlation analysis has been executed only for the Storm bot and for the Torpig bot. For the Storm bot, the inter-trace cluster correlation analysis was triggered because the intra-cluster analysis described in Section 5.1 did not produce any result, since no timing relationships were mined. On the other hand, we explicitly enabled the inter-trace cluster correlation analysis for the Torpig bot as we wanted to have a very high confidence of its behavior, considering the fact the Torpig botnet was still active at the time of the analysis. Table 2 reports information on the structure of the analyzed bot trace, e.g., push-/pull-based, and P2P, and the size of the network trace along with the number of packets and flows contained. Moreover, the table shows the number of remaining flows after the post-filter phase is finished (PF-Flows column). Next, the number of all the clusters generated during the clustering approach described in Section 4 is reported. In particular, the numbers in parenthesis represent the number of clusters created after the merging step presented in Section 5.3. Finally, the table shows the result of the cluster-based analyses described in Section 5, where information on the number of interesting clusters are reported for both the normal analysis (Single-trace column) and the optional one (Inter-trace column). The columns labeled with $\mathcal{C}_p$, $\mathcal{C}_r$, and $\mathcal{C}_i$, represent the set of clusters $\mathcal{C}_{periodic}$, $\mathcal{C}_{reconsidered}$, and $\mathcal{C}_{interesting}$, respectively.

For instance, HTTP-1 refers to an HTTP-based bot whose main goal was to crack CAPTCHA challenges from the account creation process of `hotmail.com`. The network trace generated by HTTP-1 was 2 MB in size with more than four thousands packets. These packets belonged to 206 network flows, which dropped to 151 after uncompleted connections and scan attempts were filtered out. The hierarchical clustering described in Section 4 generated 84 clusters. The intra-cluster analysis described in Section 5.1 highlighted the main behavior of the bot. In particular, the analysis identified one cluster whose flows were initiated every 20 seconds, periodically. Moreover, the inter-cluster dependency analysis pointed out the presence of 2 other clusters, which provided more insights into the bot behavior. In particular, these clusters captured the actions responsible for retrieving CAPTCHAs from `hotmail.com`, and submitting them

#	Bot	Trace (MB)	#Pkts	#Flows	#PF-Flows	# Clusters	Single-trace			Inter-trace		
							$\mathcal{C}_p$	$\mathcal{C}_r$	$\mathcal{C}_i$	$\mathcal{C}_p$	$\mathcal{C}_r$	$\mathcal{C}_i$
1	HTTP-1	2	4,285	206	151	88(84)	1	2	0	NA	NA	NA
2	HTTP-2	54	81,161	2,392	1,979	98(49)	5	14	0	NA	NA	NA
3	Torpig-1	3	16,186	3,921	1,218	113(106)	2	2	0	2	0	0
4	Torpig-2	2	15,427	3,855	577	40(37)	2	2	0	2	0	0
5	IRC-1	11	135,858	85,963	1,306	12(9)	1	4	0	NA	NA	NA
6	IRC-2	15	136,163	16,432	2,282	22(22)	1	6	0	NA	NA	NA
7	Storm-1	2.5	27,173	3,137	827	821(283)	0	0	0	0	0	2
8	Storm-2	2.3	25,853	2,960	1,023	968(218)	0	0	0	0	0	2
9	Storm-3	7.1	87,595	2,700	1,562	1,436(392)	0	0	0	0	0	1
10	Storm-GW1	42	521,097	11,192	8,132	123(71)	0	0	0	0	0	5
11	Storm-GW2	45	518,370	11,855	8,180	217(72)	0	0	0	0	0	5

Table 2: Bots Detection Models Details.

Network	Trace	# Conns	Days	IP Space	IPs Flagged	Total Alerts	Alerts/Day
Swiss ISP	77.96 TB	541,466,576	2	786,420	8	8	4.00
Greek University	128.18 TB	816,591,600	9	4,096	5	6	0.67

Table 3: False Positives

to a *C&C* server $\mathcal{S}$. The main goal of $\mathcal{S}$ was, in fact, to crack the submitted CAPTCHAs and, for this reason, the bot kept polling $\mathcal{S}$ asking for the textual representation of the submitted CAPTCHA.

Similarly to the above description, our cluster-based analysis pointed out clusters showing timing-relationships for the majority of the other bots shown in Table 2. For instance, HTTP-2 behavior is described by the presence of five clusters whose flows were showing periodically every 948 seconds. Beside them, 14 more clusters were retrieved by the inter-cluster dependency analysis. All the communications involved in these clusters were directed toward `google.com` hosts. Unfortunately, we could not get more insights into the bot malicious behavior as the majority of the communications involved in those clusters were carried over SSL. Nonetheless, our analysis was able to generate a network-based behavioral detection model.

In a similar fashion, the Torpig bot [9] periodically contacted two *C&Cs*, as highlighted by our cluster-based analysis. In particular, the bot contacted a Torpig *C&C* and a Mebroot *C&C* every 1200 and 7200 seconds, respectively. The Torpig *C&C* was used as a repository for sensitive information stolen by the bot, while the Mebroot *C&C* provided a way for the botherder to update, install, and uninstall the malware itself.

The intra-cluster analysis ran on the bot labeled IRC-1 and IRC-2 highlighted, for each of them, one cluster whose flows exhibited a periodic behavior. In particular, these flows were responsible for PING/PONG IRC messages sent by the IRC *C&C* server to check for the presence of the bot, its responsiveness, and general connectivity. As in the previous cases, the clusters resulting from the optional inter-cluster dependency analysis provided more insights on the behavior of the bots (e.g., joining specific channels, setting channel topic, and sending private messages).

Of all the bot examined so far, our analysis was able to extract some periodic behavior. The only exception is represented by the P2P-based Storm bot [18]. We analyzed five Storm traces, three of which, i.e., Storm-1, Storm-2, and Storm-3, belonged to bots installed on machines behind NATs, while the remaining two, i.e., Storm-GW1, and Storm-GW2, belonged to bots on hosts with public IPs. Our intra-cluster timing analysis did not find any apparent timing relationship. However, the inter-trace cluster correlation analysis was able to find similar clusters in all the traces. In particular, *all* the Storm bots shared one cluster possibly representing a form of weak authentication in which the bots engaged in the beginning. As a consequence, our analysis produced a model able to detect hosts infected by instances of the Storm bot.

To further evaluate the detection capability of our models, we randomly selected 64 traces out of 308 (i.e., 20%), to build a training set $\mathcal{T}$. Then, for each trace in $\mathcal{T}$, a network behavior detection model was automatically generated, as outlined in Section 5. Subsequently, we deployed a Bro sensor, equipped with these models, to determine their detection capability, and, in particular, the detection rate over all the remaining traces not included in the training set. In particular, this procedure was performed four times (four-fold cross validation). The results show that our models reported a bot infection for 50% of the analyzed traces. This corresponds to 154 detection over the entire set of 308 network traces. At a first glance, this number seems low. The reason has to be found in the way models are generated. In fact, for some of the models, networking and timing properties produce well-defined ranges that do not overlap easily with those defined by other models. However, this restriction can be relaxed by tolerating more false positives.

To evaluate the false positive rate caused by our approach, we deployed our models on a Bro sensor at two different network sites. In particular, we deployed one Bro sensor at a Swiss ISP network, and one sensor at a Greek university network. In Switzerland, our system monitored a populated network (789K IPs) for 2 days (off-line traffic). In Greece, our system monitored a medium-populated network (4K IPs) for 9 days (on-line traffic). Table 3 summarizes the main results of our false positive evaluation. It can be observed that only few false positives are raised over a total period of 9 days. In particular, only 13 IPs were flagged as infected by bots and only 14 alerts were raised. Most of these false alarms were caused by periodic communications initiated by email clients toward POP3 servers. Of course, it is rather difficult to state whether these alarms were truly all false positives and not true ones as we did not have access to the offending hosts. Nonetheless, it is reasonable to accept this fact as both networks are well-maintained and bot infections, in these networks, are very rare.

9 Conclusions

Botnets have become a serious security issue that threatens the confidentiality and integrity of users' data.

In this paper we present a system that monitors network traffic to detect bots. To this end, we propose an analysis that highlights cluster-based properties of flows deemed to be similar with respect to a given set of network features and timing relationships. Our analysis automatically generates network behavior models that detect individual infected hosts, with very few false positive. Moreover, traffic content is not inspected, which makes our approach robust when bots use encrypted communication. In addition, our approach does not require to observe noisy behavior, and it is thus suitable for detecting bots whose goal is information theft. Our tool automatically generates models for different bot families, including HTTP-based, IRC-based, and P2P-based bots, that produce very few false positive. This suggests our network behavior detection models can be deployed in real-world settings.

References

[1] Ulrich Bayer, Christopher Kruegel, and Engin Kirda. Anubis: Analyzing Unknown Binaries. In *http://anubis.iseclab.org/*, 2008.

[2] Ulrich Bayer, Paolo Milani Comparetti, Clemens Hlauscheck, Christopher Kruegel, and Engin Kirda. Scalable, Behavior-Based Malware Clustering. In *16th Symposium on Network and Distributed System Security (NDSS)*, 2009.

[3] James R. Binkley. An algorithm for anomaly-based botnet detection. In *SRUTI '06*, pages 43–48, 2006.

[4] Mihai Christodorescu and Somesh Jha. Testing malware detectors. In *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004)*, pages 34–44, Boston, MA, USA, July 2004. ACM Press.

[5] Mihai Christodorescu, Somesh Jha, Sanjit A. Seshia, Dawn Song, and Randal E. Bryant. Semantics-aware malware detection. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy (Oakland 2005)*, pages 32–46, Oakland, CA, USA, May 2005.

[6] Evan Cooke, Farnam Jahanian, and Danny McPherson. The zombie roundup: understanding, detecting, and disrupting botnets. In *SRUTI'05: Proceedings of the Workshop on Steps to Reducing Unwanted Traffic on the Internet*, pages 6–6, Berkeley, CA, USA, 2005. USENIX Association.

[7] Morris H. Degroot and Mark J. Schervish. *Probability and Statistics (2nd Edition)*. Addison Wesley, 2001.

[8] Manuel Egele, Christopher Kruegel, Engin Kirda, and Heng Yin. Dynamic spyware analysis. In *In Proceedings of the 2007 Usenix Annual Conference (Usenix 07*, 2007.

[9] E. Florio and K. Kasslin. Your computer is now stoned (...again!). *Virus Bulletin*, April 2008.

[10] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An Inquiry into the Nature and Causes of the Wealth of Internet Mi screants. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, 2007.

[11] Felix C. Freiling, Thorsten Holz, and Georg Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In *In Proceedings of 10 th European Symposium on Research in Computer Security, ESORICS*, pages 319–335, 2005.

[12] Jan Goebel and Thorsten Holz. Rishi: identify bot contaminated hosts by irc nickname evaluation. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 8–8, Berkeley, CA, USA, 2007. USENIX Association.

[13] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. BotMiner: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th USENIX Security Symposium (Security'08)*, 2008.

[14] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of the 16th USENIX Security Symposium (Security'07)*, August 2007.

[15] Guofei Gu, Junjie Zhang, and Wenke Lee. BotSniffer: Detecting botnet command and control channels in network traffic. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, February 2008.

[16] P. Gutmann. The Commercial Malware Industry. In *Proceedings of the DEFCON conference*, 2007.

[17] Jiawei Han. Efficient mining of partial periodic patterns in time series database. In *Proc. Int. Conf. on Data Engineering*, pages 106–115, 1999.

[18] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: a case study on storm worm. In *LEET'08: Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*, pages 1–9, Berkeley, CA, USA, 2008. USENIX Association.

[19] Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-scale botnet detection and characterization. In *HotBots'07: Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, pages 7–7, Berkeley, CA, USA, 2007. USENIX Association.

[20] Andrea Lanzi, Monirul Sharif, and Wenke Lee. K-tracer: A system for extracting kernel malware behavior. In *the 16th Annual Network and Distributed System Security Symposium (NDSS'09)*, 2009.

[21] Sheng Ma and Joseph L. Hellerstein. Mining partially periodic event patterns with unknown periods. *Data Engineering, International Conference on*, 0:0205, 2001.

[22] Lorenzo Martignoni, Elizabeth Stinson, Matt Fredrikson, Somesh Jha, and John C. Mitchell. A Layered Architecture for Detecting Malicious Behaviors. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection, RAID, Cambridge, Massachusetts, U.S.A.*, Lecture Notes in Computer Science. Springer, September 2008.

[23] Vern Paxson. Bro: A system for detecting network intruders in real-time. In *Computer Networks*, pages 2435–2463, 1999.

[24] Moheeb A. Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *IMC '06: Proceedings of the 6th ACM SIGCOMM on Internet measurement*, pages 41–52. ACM Press, 2006.

[25] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Chris Kruegel, and Giovanni Vigna. Your botnet is my botnet: Analysis of a botnet takeover. Technical report, University of California, Santa Barbara, April 2009.

[26] W. T. Strayer, R. Walsh, C. Livadas, and D. Lapsley. Detecting botnets with tight command and control. In *Proceedings of the 31st IEEE Conference on Local Computer Networks*, pages 195–202, 2006.

[27] Rui Xu and D. Wunsch. Survey of clustering algorithms. *Neural Networks, IEEE Transactions on*, 16(3):645–678, 2005.

[28] Ting-Fang Yen and Michael K. Reiter. Traffic aggregation for malware detection. In *DIMVA '08: Proceedings of the 5th international conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 207–227, Berlin, Heidelberg, 2008. Springer-Verlag.

[29] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. Panorama: capturing system-wide information flow for malware detection and analysis. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 116–127, New York, NY, USA, 2007. ACM.