BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection

Florian Tegeler
University of Göttingen
tegeler@cs.uni-goettingen.de

Xiaoming Fu
University of Göttingen
fu@cs.uni-goettingen.de

Giovanni Vigna
UC Santa Barbara
vígna@cs.ucsb.edu

Christopher Kruegel
UC Santa Barbara
chris@cs.ucsb.edu

ABSTRACT

Bots are the root cause of many security problems on the Internet, as they send spam, steal information from infected machines, and perform distributed denial-of-service attacks. Many approaches to bot detection have been proposed, but they either rely on end-host installations, or, if they operate on network traffic, require deep packet inspection for signature matching.

In this paper, we present BOTFINDER, a novel system that detects infected hosts in a network using only high-level properties of the bot's network traffic. BOTFINDER does not rely on content analysis. Instead, it uses machine learning to identify the key features of command-and-control communication, based on observing traffic that bots produce in a controlled environment. Using these features, BOTFINDER creates models that can be deployed at network egress points to identify infected hosts. We trained our system on a number of representative bot families, and we evaluated BOTFINDER on real-world traffic datasets – most notably, the NetFlow information of a large ISP that contains more than 25 billion flows. Our results show that BOTFINDER is able to detect bots in network traffic without the need of deep packet inspection, while still achieving high detection rates with very few false positives.

Categories and Subject Descriptors

C.2.0 [**General**]: Security and Protection

Keywords

Malware Detection, Security, NetFlow Analysis

1. INTRODUCTION

Many security problems on today's Internet such as spam, distributed denial-of-service (DDoS) attacks, data theft, and click fraud are caused by malicious software running undetected on end-user machines. The most efficient, and arguably, most relevant kind of such malware are *bots*. The malicious software components are coordinated over a *command and control (C&C)* channel by a single entity – called the *botmaster* – and form a *botnet* [6,13,29] to carry out a number of different criminal activities. Consequently, defenses against malware infections are a high priority, and the identification of infected machines is the first step to purge the Internet of bots.

Acknowledging the limitations of traditional host-based malware detection, such as anti-virus scanners, network-based bot detection approaches are increasingly deployed for complementary protection. Network devices provide a number of advantages, such as the possibility to inspect a large number of hosts without the need for any end-point software installation.

Recently, a trend toward smaller botnets [6] and a shift [11] in malware development from a for-fun activity to a for-profit business was observed. This introduces very stealthy bots with encrypted *C&C* communication and we derive three core design goals for network based solutions to capture such bots: (a) The system should be able to detect individual bot infections. (b) The system should rely only on network-flow information to be resilient to encrypted traffic, and (c) should work for stealthy bots that do not send spam or carry out DoS attacks but steal sensitive data (e.g., credit cards or login credentials).

In this paper, we present BOTFINDER, a system that detects individual, bot-infected machines by monitoring their network traffic. BOTFINDER leverages the observation that *C&C* connections associated with a particular bot family follow certain regular patterns. That is, bots of a certain family send similar traffic to their *C&C* server to request commands, and they upload information about infected hosts in a specific format. Also, repeated connections to the command and control infrastructure often follow certain timing patterns.

BOTFINDER works by automatically building multi-faceted models for *C&C* traffic of different malware families. To this end, we execute bot instances that belong to a single family in a controlled environment and record their traffic. In the next step, our system extracts features related to this traffic and uses them to build a detection model. The detection model can then be applied to unknown network traffic. When traffic is found that matches the model, we flag the host responsible for this traffic as infected.

Due to its design, BOTFINDER offers a combination of salient properties that sets it apart from previous work and fulfills the aforementioned design goals. Our solution is able to detect individual bot infections and does not correlate activity among multiple hosts during the detection phase as, for example, BotSniffer [18], BotMiner [16], or TAMD [37]. Moreover, such systems rely on noisy activity, such as spamming and DoS activity (for example called A-

Co-NEXT'12, December 10-13, 2012, Nice, France.

Plane in BotMiner) which prevents the detection of stealthy bots. Yet, many existing systems [15, 17, 36] allow the detection of individual infections, but they use deep packet inspection. In contrast, BOTFINDER requires only high-level (NetFlow-like [5]) information about network connections; it does not inspect payloads. Thus, it is resilient to the presence of encrypted bot communication, and it can process network-level information (e.g., NetFlow) that is typically easier to obtain than full-packet dumps (because of the privacy concerns of network operators).

We evaluated our approach by generating detection models for a number of botnet families. These families are currently active in the wild, and make use of a mix of different infection and *C&C* strategies. Our results show that BOTFINDER is able to detect malicious traffic from these bots with high accuracy. We also applied our detection models to traffic collected both on an academic computer laboratory network and a large ISP network (with tens of billions of flows), demonstrating that our system produces promising results with few false positives.

In summary, this paper makes the following contributions:

- We observe that *C&C* traffic of different bot families exhibits regularities (both in terms of traffic properties and timing) that can be leveraged for network-based detection of bot-infected hosts. Being independent of packet payloads, our detection approach can handle encrypted or obfuscated traffic.
- We present BOTFINDER, a learning-based approach that automatically generates bot detection models. To this end, we run bot binaries in a controlled environment and record their traffic. Using this data, we build models of characteristic network traffic features.
- We develop a prototype of BOTFINDER, and we show that the system is able to operate on high-performance networks with hundreds of thousands of active hosts and Gigabit throughput in real time. We apply BOTFINDER to real traffic traces and demonstrate its high detection rate and low false positive rate. Additionally, we show that BOTFINDER outperforms existing bot detection systems and discuss how BOTFINDER handles certain evasion strategies by adaptive attackers.

2. SYSTEM OVERVIEW

BOTFINDER detects malware infections in network traffic by comparing statistical features of the traffic to previously-observed bot activity. Therefore, BOTFINDER operates in two phases: a *training* phase and a *detection* phase. During the training phase, our system learns the statistical properties that are characteristic of the command and control traffic of different bot families. Then, BOTFINDER uses these statistical properties to create models that can identify similar traffic. In the detection phase, the models are applied to the traffic under investigation. This allows BOTFINDER to identify potential bot infections in the network, even when the bots use encrypted *C&C* communication.

Figure 1 depicts the various steps involved in both phases: First, we need to obtain input for our system. In the training phase, this input is generated by executing malware samples in a controlled environment (such as Anubis [2], BitBlaze [30], CWSandbox [35], or Ether [8]) and capturing the traffic that these samples produce. In the second step, we reassemble the flows[1] in the captured traffic; a step that can be omitted when NetFlow data is used instead of full packet captures. In the third step, we aggregate the flows in traces – chronologically-ordered sequences of connections between two

[1]We will use the words *flow* and *connection* interchangeably in this paper.

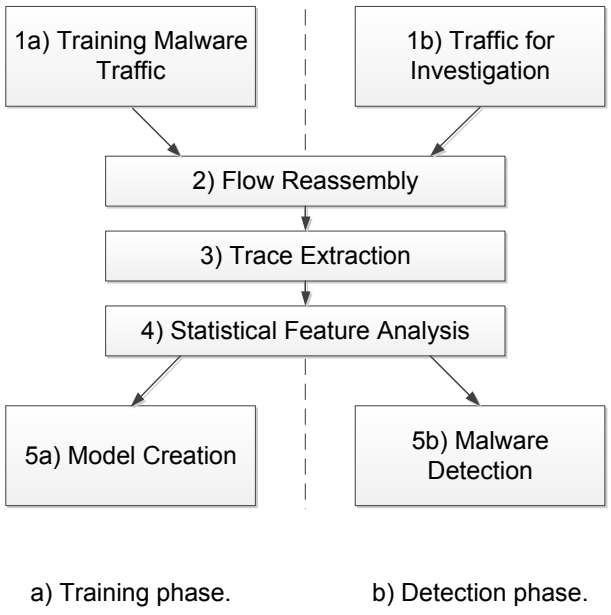


Figure 1: General architecture of BOTFINDER.

IP addresses on a given destination port. BOTFINDER then extracts five statistical features for each trace in the forth step. These statistical features are the average *time* between the start times of two subsequent flows in the trace, the average *duration* of a connection, the *number of bytes* on average transferred to the source, the number of bytes on average transferred to the destination, and a *Fourier Transformation over the flow start times* in the trace. The latter allows us to identify underlying frequencies of communication that might not be captured by using simple averages. Finally, in the fifth step, BOTFINDER leverages the aforementioned features to build models. During model creation, BOTFINDER clusters the observed feature values. Each feature is treated separately to reflect the fact that we did not always observe correlations between features: For example, a malware family might exhibit similar periodicity between their *C&C* communications, but each connection transmits a very different number of bytes. The combination of multiple clusters for each of a bot's features produces the final malware family model.

When BOTFINDER works in the detection phase, it operates on network traffic and uses the previously-created models for malware detection.

It is important to note that we do not rely on any payload information of the traffic for the whole process, but we work on the statistical properties exhibited by the command and control communication only.

3. SYSTEM DETAILS

This section provides more details on how BOTFINDER and the previously-mentioned five steps work.

3.1 Input Data Processing

The input for BOTFINDER is either a traffic capture or NetFlow data. During the training phase, malware samples are executed in a controlled environment, and all network traffic is recorded. In this step, it is important to correctly classify the malware samples

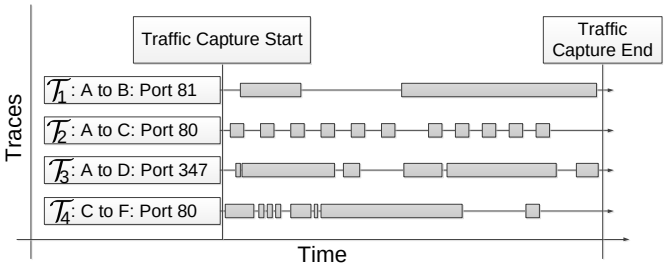


Figure 2: Traces with different statistical behaviors.

so that different samples of the same malware family are analyzed together. Our classification is based on the output of various antivirus scanners executed via VirusTotal[2] and on the results of behavioral similarity analysis in Anubis [1]. Of course, incorrectly classified samples are possible. This might affect the quality of the produced models. However, as explained later, BOTFINDER tolerates a certain amount of noise in the training data.

3.2 Flow Reassembly

If NetFlow data is available, BOTFINDER directly imports it, otherwise, we reassemble flows from captured packet data. For each connection, properties such as start and end times, the number of bytes transferred in total, and the number of packets is extracted. As a final result of this reassembly step, our system yields aggregated data similar to NetFlow, which is the industry standard for traffic monitoring and IP traffic collection. For all further processing steps, BOTFINDER only operates on these aggregated, content-agnostic data.

3.3 Trace Extraction

Traces are an important concept in BOTFINDER: A trace $\mathcal{T}$ is a sequence of chronologically-ordered flows between two network endpoints. Figure 2 illustrates different shapes of traces showing start times and durations of flows. For example, the trace $\mathcal{T}_2$ from A to C on port 80 shows a highly regular behavior. The regularity in $\mathcal{T}_2$ allows BOTFINDER to extract the recurrence and statistical features over all flows of $\mathcal{T}_2$. Here, the roughly constant distance between two flows and the similar duration of communication allows for an accurate description of the whole trace, using the average time distance between flows and their average duration.

To obtain meaningful statistical data, we require at least a minimal number of connections $|\mathcal{T}|_{min}$ (typically between 10 and 50) in a trace $\mathcal{T}$. This minimal length requirement is consistent with the fact that command-and-control traffic consists of multiple connections between the infected host and the *C&C* server.

A challenge in performing fully-automated analysis of malware samples is to distinguish between traces that correspond to actual command and control interactions and traces that are just additional "noise." Many bots connect to legitimate sites, for various different reasons, such as checking for network connectivity, checking for the current time, or for sending spams. Some bot variants even deliberately create noisy benign traffic to legitimate websites to cloak their own *C&C* communication [9, 10, 26] and to counter automatic signature generation systems. We use two ways to filter the traffic and identify the relevant traffic traces: First, we easily whitelist common Internet services such as Microsoft Update and Google. In addition, if available, we leverage third-party knowledge and compare our training traffic (whenever un-encrypted) to known signatures or special communication patterns. Moreover, we compare the destination IP addresses to a list of known *C&C* servers. Another, more advanced and automated technique that allows identification of previously unknown *C&C* servers is JACKSTRAWS [21], an approach that leverages additional system call information from the bot sample execution. Distinguishing between *C&C* traffic and unrelated traces allows models to capture only characteristic bot traffic. Interestingly, when traffic that is not related to *C&C* connections is included into the model generation process, the resulting models are typically of low confidence (as shown later). As a result, they have little impact on the detection results.

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

3.4 Feature Extraction

After trace generation, BOTFINDER processes each trace to extract relevant statistical features for subsequent trace classification. We focus on the following five features:

- The average **time interval** between the start times of two subsequent flows in the trace. The botmaster has to ensure that all bots under his control receive new commands and updates frequently. Often, communication from the *C&C* server to the bots, following a push model, is impossible. The reason is that many infected hosts in private networks are behind NAT boxes or not registered with the *C&C* server yet. We assume that most bots use a constant time interval between *C&C* connections (or a random value within a certain, specific time interval). This leads to detectable periodicity in the communication. For the communication pattern, the botmaster has to balance the scalability and agility of his botnet with the increasing risk of detection associated with an increasing number of *C&C* server connections. As mentioned before, some bot variants already open random, benign connections [9, 10, 26] to distract signature generation and malware detection systems. Other approaches, such as "connect each day at time X" also suffer from issues like the requirement of synchronization between the bots' host clocks. Nevertheless, malware authors might craft their bots explicitly to not show periodic behavior. As we discuss in more detail in Section 6, mimicking random, benign traffic is hard and often detectable. Based on our observations working with different malware families, we found that a significant fraction of current malware follows our assumption and exhibit loosely periodic *C&C* communication.
- The average **duration** of connections. As bots often do not receive new commands, most of the communication consists of a simple handshake: The bot requests new commands and the *C&C* server responds that no new commands have been issued. Thus, we expect that the durations for most flows in a *C&C* trace are similar.
- The average number of **source bytes** and **destination bytes** per flow. By splitting up the two directions of communication using source and destination bytes, we are able to separate the request channel from the command transmission. That is, the request for an updated spam address list might always be of identical size, while the data transferred from the *C&C* server, containing the actual list, varies. As a result, a *C&C* trace might contain many flows with the same number of source bytes. Similar considerations apply to the destination bytes – for example, when the response from the *C&C* server has a fixed format.
- The **Fast Fourier Transformation** (FFT) over a binary sampling of the *C&C* communication with the goal to detect underlying communication regularities. In this step, we sample our trace like a binary signal by assigning it to be 1 at each connection start, and 0 in-between connections. To calculate a high-quality FFT, we

used a sampling interval of $1/4$th of the smallest time interval in the trace, which ensures that we do not undersample. However, if the distance between two flows is extremely small and large gaps occur between other flows of the trace, this sampling method can lead to a significant amount of data points. In such cases, we limit the length of our FFT trace to $2^{16} = 65,536$ datapoints and accept minor undersampling. We chose this value as the FFT is fastest for a length of power of two, and, with this value, only a few datapoints in our experiments were (under)sampled as a single one. More precisely, for the observed *C&C* traces, 18% showed undersampling, which resulted in a median of only 1% of the start times that were sampled together.

In the next step, we compute the Power Spectral Density (PSD) of the Fast Fourier Transformation over our sampled trace and extract the most significant frequency. The FFT peaks are correlated with time periodicities and resistant against irregular large gaps in the trace (as we will show in Section 6). We observed the introduction of gaps in the wild for bots in which communication with the *C&C* server is periodic and then pauses for a while. When malware authors randomly vary the *C&C* connection frequency within a certain window, the random variation lowers the FFT peak. However, the peak remains detectable and at the same frequency, enabling the detection of the malware communication.

3.5 Model Creation (Training)

We create models via clustering of the five features: average time, average duration, average source bytes, average destination bytes, and the FFT. We process the dataset for each feature separately, as we observed malware behavior with non-correlated features. As an example of such behavior, two versions of a bot might connect to different versions of *C&C* servers C_1 and C_2 and transfer different amounts of bytes, depending of their version. Nevertheless, these two bot versions might still follow the same communication periodicity pattern.

After clustering, we typically observe a number of rather large clusters that contain the –suspected – actual malware-specific behavior. In addition, there are often some smaller clusters with more diverse data (lower clustering quality) and even individual traces present. These small clusters are typically related to non-*C&C* traffic, and our analysis drops them. A final model for a malware family contains five sets of clusters, one set for each feature. A set of clusters for a feature characterizes the expected values for this feature. In human terms, a model can be understood as:

> *An average interval between connections of 850 or 2,100 seconds, a transfer of 51kB to the source, 140 bytes to the destination, a flow duration of 0.2 or 10 seconds, and a communication frequency of around 0.0012Hz or 0.04Hz indicate a Dedler infection.*

To cluster the trace-features for a bot family, we use the CLUES (CLUstEring based on local Shrinking) algorithm [34], which allows non-parametric clustering without having to select an initial number of clusters. In short, CLUES iteratively applies gravitational clustering [23] and selects a number of clusters that complies best to a cluster strength measure such as the Silhouette index by Kaufman and Rousseuw [22].

To confirm the applicability of CLUES for BOTFINDER, we applied CLUES to our datasets and compared its clustering results with the results obtained by running the well-known k-means algorithm [20]. For our datasets, the fully automated, non-supervised CLUES algorithm typically showed the same results as manually-supervised k-means clustering. In some cases, we even found better cluster formation than with k-means. This demonstrates that CLUES is a good candidate for our clustering scenario.

After calculating cluster centers and members, we judge the quality of each individual cluster using a quality rating function. As we tend to trust large clusters with highly similar values over smaller, more diverse clusters, we relate the standard deviation sd of the cluster with its mean c and calculate the fraction sd/c. This fraction is then used as part of an exponentially decreasing quality rating function $q_{cluster} = \exp(-\beta \cdot \frac{sd}{c})$ with a control factor β, empirically set to 2.5.

The average $q_{cluster}$ over all clusters is a measure of the overall input trace similarity. A high average cluster quality indicates that many binary samples generated highly similar traces that yield similar extracted features. If the traces are more diverse, more clusters of lower quality exists, which is, however, not necessarily a bad sign: Imagine a malware family that has a fixed interval in its periodic *C&C* communication but tries to evade detection by adding artificial, random traffic. As described in Section 3.3, we are trying to extract the relevant *C&C* communication from the traffic generated by the malware sample, yet this process is error prone. Throughout the clustering process, the fixed interval *C&C* communication emitted by most samples is clustered in a high quality cluster whereas the random traffic clusters very bad and generates very loose clusters with high standard deviation. Such clusters have a low quality and reduce the average cluster quality, however the cluster that captured the actual *C&C* communication is still of high quality and expresses the relevant malware behavior well.

3.6 Model Matching (Detection)

To check whether a trace $\mathcal{T}$ matches a given model M, we compare each statistical feature of this trace with the model's clusters that were generated in the previous steps. If, for example, the trace's average time property lies in one of M's clusters for this feature, we count this as a "hit" and increase a score γ_M. The amount by which the score γ_M is increased depends on the quality of the cluster and the quality of the (feature of the) trace. These "qualities" reflect the uncertainties inherited by trace collection and feature extraction during the previous steps. Additionally, we consider how "tight" the feature of a trace (for example, the average time) is regarding its periodicity. In general, for higher cluster qualities and tighter traces, γ_M is increased more. More precisely, we add $q_{cluster} \cdot \exp\{-\beta \frac{sd_{trace}}{avg_{trace}}\}$, with β again set to 2.5, to γ_M for all values that hit a cluster by matching its cluster center $\pm$ two times the cluster's standard deviation. The limitation to this range is primarily motivated to optimize processing speed. Mathematically the described exponential scoring function decreases very quickly, therefore comparison of any value with the cluster center would contribute nearly 0 to γ_M for values off more than twice the standard deviation.

To allow hits in multiple models M_1 and M_2 for different bot families, we maintain a γ_M for each model. Note that clusters of low quality, which are often introduced as artifacts of the training data (from traffic unrelated to *C&C* communication), only lead to a small increase of γ_M. In this fashion, the system implicitly compensates for some noise in the training data.

Finally, the highest γ is compared to a global pre-specified acceptance threshold a, which has to be set by the BOTFINDER administrator. If $\gamma > a$, the model is considered to have matched, and BOTFINDER raises an alarm. To reduce false positives and not rely on a single feature alone, we allow the user to specify a minimal number of feature hits h. That is, in addition to the requirement $\gamma > a$, the trace has to have matches for at least h features. This rule avoids accidental matches solely based on a single feature. For

Table 1: Malware families used for training. A high cluster quality indicates a low standard deviation within the clusters.

Family	Samples	Total Traces	Cluster Quality [0,1]
Banbra	29	29	0.99
Bifrose	33	31	0.52
Blackenergy	34	67	0.57
Dedler	23	46	0.76
Pushdo	55	106	0.49
Sasfis	14	14	0.88
Average	32	49	0.70

example, consider a trace with features "average time" and "FFT" that match the model clusters very well (so that the condition $\gamma > a$ is satisfied). By setting $h = 3$, BOTFINDER requires that an additional feature, such as the "average duration" or one of the byte transmission features, hits its respective model to actually raise an alarm.

4. TRAINING

We trained BOTFINDER on six different malware families that are a representative mix of families currently active and observed in the wild. More precisely, we picked active malware samples that we observed in Anubis within a window of 30 days in June 2011. Anubis receives and analyzes tens of thousands of samples every day. This selection process ensures that our system operates on malware that is active and relevant.

For each family, we executed on average 32 samples in a Windows XP VM in our controlled, Virtualbox[3]-based environment for one to two days and captured all network traffic. The virtual machine runs a real OS, is connected to the Internet, and contains realistic user data. Of course, we restricted SPAM and DoS attempts. The malware families used for training are:

- **Banbra**: A Trojan horse/spyware program that downloads and installs further malware components.
- **Bifrose** (also represented in Trojan variants called *Bifrost*): A family of more than 10 variants of backdoor Trojans that establish a connection to a remote host on port 81 and allow a malicious user to access the infected machine. It periodically sends the hosts status information and requests new commands.
- **Blackenergy**: A DDoS bot that communicates through HTTP requests. Blackenergy's current version 2 increased detection countermeasures, such as strong encryption and polymorphism.
- **Dedler**: A classical spambot that evolved through different versions from a simple worm that spreads through open fileshares to an advanced Trojan/spambot system. Whereas initial versions appeared already in 2004, recent versions are still prevalent and active and in the traffic traces analyzed, massive spam output was observed.
- **Pushdo** (also known as *Pandex* or *Cutwail*): An advanced DDoS and spamming botnet that is active and continuously evolving in the wild since January 2007.
- **Sasfis**: A Trojan horse that spreads via spam and allows the remote control of compromised machines. Following typical bot behavior, the *C&C* channel is used to transfer new commands or download additional malware to the computer.

Table 1 shows the detailed distribution of malware samples and associated malware traces. The "Cluster Quality" column reflects the quality rating function's results. A high value implies close clusters (low standard deviation) and indicates that our core assumption holds: Different binaries of the same malware family produce similar *C&C* traffic, and this traffic can be effectively described using clustering techniques. However, as aforementioned, a low average cluster quality does not necessarily reflect ineffective capture of a malware's behavior. For example, the largest clusters for each feature in the Pushdo model have a high quality > 0.9. However, many small clusters with low qualities reduce the overall cluster quality. Still the large, high quality clusters give a good representation of the Pushdo behavior. For Bifrose, the clusters are in general more diverse due to higher variances in the traces generated by the malware binaries and the model has to be considered weaker than for the other bot families. For the clustering we used the default values as described in Chang et al. [4] and obtained on average 3.14 (median 3) clusters per feature for each family.

5. EVALUATION

To evaluate BOTFINDER, we performed a number of experiments on two real-world datasets (see Table 2 for a summary): The first dataset, *LabCapture*, is a full packet capture of 2.5 months of traffic of a security lab with approximately 80 lab machines. According to the lab policy, no malware-related experiments should have been executed in this environment, and the *LabCapture* should consist of only benign traffic. As we have the full traffic capture, we are also able to manually verify reported infections. The second dataset, *ISPNetFlow*, covers 37 days of NetFlow data collected from a large network. The captured data reflects around 540 Terabytes of data or 170 Megabytes per second of traffic. We are aware that we do not have ground truth for the second network dataset, as we lack the underlying, full traffic capture that would be required for full content inspection. Nevertheless, we can compare our identified hits to known malware IP blacklists, identify clusters of infected machines, and judge the usability of our approach for the daily operation of large networks.

After developing a prototype implementation, we performed the following experiments (for a detailed description please refer to the respective subsections):

- A cross-validation experiment based on our ground truth training data and the *LabCapture* dataset: In short, the training data is split into a training set and a detection set. The latter is then mixed with all traces from the *LabCapture* data that should not contain bot traces. After BOTFINDER has learned the bots' behavior on the training set, we analyzed the detection ratio and false positives in the dataset that contained both the remaining known malicious traces and the *LabCapture* data. **[Section 5.2]**
- Comparison to related work: In our case, the most relevant related work is the well-known, packet-inspection-based system *BotHunter* [17]. We performed all experiments on a set of a fraction of ground truth *C&C* traces mixed with the *LabCapture* dataset. **[Section 5.3]**
- *ISPNetFlow* analysis: We trained BOTFINDER on all training traces and ran it on the *ISPNetFlow* dataset in daily slices. We investigated the identified malicious traces and compared it to blacklisted malicious *C&C* server IPs. **[Section 5.4]**

5.1 Implementation and Performance

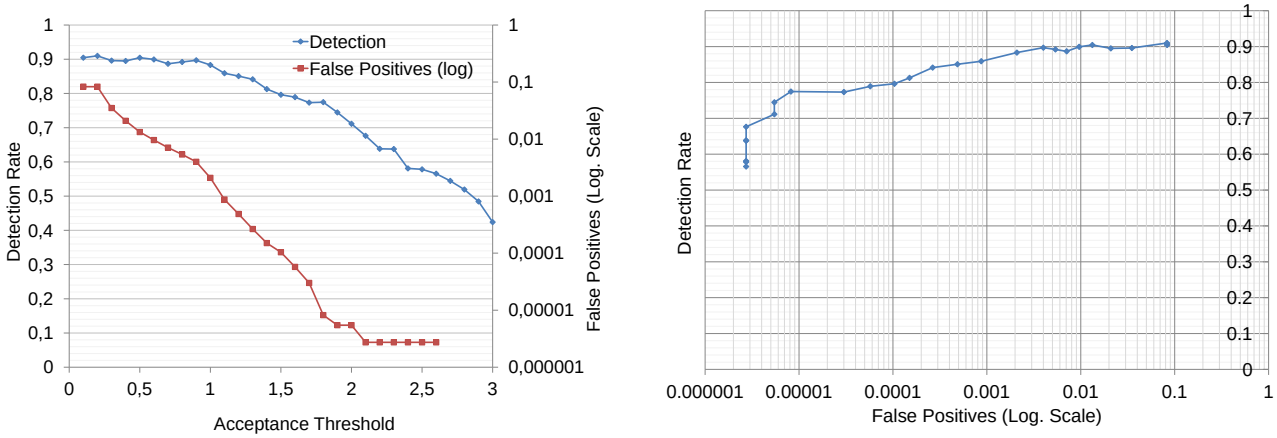
We implemented BOTFINDER in Python. For flow reassembly from full packet data captures, we utilized the intrusion detection system Bro [25]. Our implementation also operates on FlowTools[4]-

[3] `http://www.virtualbox.org`

[4] `http://www.splintered.net/sw/flow-tools/`.

Table 2: Evaluation Datasets

Name	Traffic	Internal Hosts	Concurrently Active	Start Time	Length	Connections	Long Traces
LabCapture	$\approx$ 3.6 TB	$\approx$80	$\approx$60	2011-05-04	84 days	$\approx 64.3 \cdot 10^6$	$\approx$ 39k
ISPNetFlow	$\approx$ 540 TB	$\approx$1M	$\approx$250k	2011-05-28	37 days	$\approx 2.5 \cdot 10^{10}$	$\approx$ 30M



(a) (b)

Figure 3: Detection rate and false positives of BOTFINDER in cross validation experiments.

compressed NetFlow data. BOTFINDER is able to process network information, even for high-speed Gigabit networks, in real time. In our setup, we used a lab machine equipped with an Intel Core i7 CPU with eight cores and 12GB of RAM. Using this setup, we were able to process half a billion NetFlow records or $\approx$ 33 GB of stored NetFlows, which reflected approximately one day of traffic in the *ISPNetFlow* dataset network, in about three hours.

As the feature extraction for a given trace $\mathcal{T}_A$ does not depend on any features of another trace $\mathcal{T}_B$, we were able to distribute the computational load of statistical analysis. Our implementation supports remote machines that receive traces to analyze, and reply with the aggregated features for these traces. On the worker site, these thin clients use Python's multiprocessing library to fully utilize all cores, especially for the FFT sampling and calculations, which we performed in the statistical computing environment *R* [28]. During our analysis of the large *ISPNetFlow* dataset, we used eight remote workers from our cluster (48 cores in total) for data processing and the aforementioned lab machine for data reading and analysis.

As our system allows an arbitrary number of worker CPUs, the primary bottleneck is reading and writing back to disk. Roughly, BOTFINDER requires 1.2 seconds to read 100,000 lines of FlowTools compressed NetFlow and 0.8 seconds to read 100,000 lines of Bro IP flow output. Each line represents a connection dataset. If a full packet traffic capture is analyzed, it is necessary to add the processing time for Bro to pre-process the output and create the IP flow file. Please note that all data handling and most of the processing is performed directly in Python; a native fast programming language can be expected to boost the processing performance significantly.

We used pyflowtools `http://code.google.com/p/pyflowtools/` for handling FlowTools files in Python.

5.2 Cross-Validation

To determine the detection capabilities of BOTFINDER we performed a cross-validation experiment based on our labeled ground truth training data and the *LabCapture* dataset. Both datasets were collected in the same network environment to ensure similarly capable connectivity settings. For each varying acceptance threshold a, we performed 50 independent cross-validation runs as follows:

1: We split our ground truth malware dataset (from Table 1) into a training set $\mathcal{W}$ (70% of the traces) and a detection set $\mathcal{D}$ (the remaining 30%).

2: We mixed $\mathcal{D}$ with the traces from the *LabCapture* dataset and assumed the *LabCapture* dataset to be completely infection-free, and therefore a reasonable dataset to derive the false positive ratio of BOTFINDER.

3: Further, we trained BOTFINDER and created models on the bot behavior exhibited in the traces in $\mathcal{W}$.

4: Finally, we applied these models to the mixed set combined from $\mathcal{D}$ and the *LabCapture* dataset.

The analysis is performed on a per-sample level, as we have the information of which malware binary generated a specific trace. More precisely, if one trace of a sample is correctly identified by a trace match, we count the entire sample as correctly identified; if a trace of a given malware is classified as a different malware, we consider this match as a false positive.

Figure 3 shows the detection rates for $a \in [0, 3]$ and $h = 3$. Very low acceptance thresholds yield high detection rates of above 90%, but with high false positives. For example, the false positive rate was greater than 1% for $a \leq 0.6$. As can be seen in Figure 3(a), the false positive rate decreases exponentially (near linear in

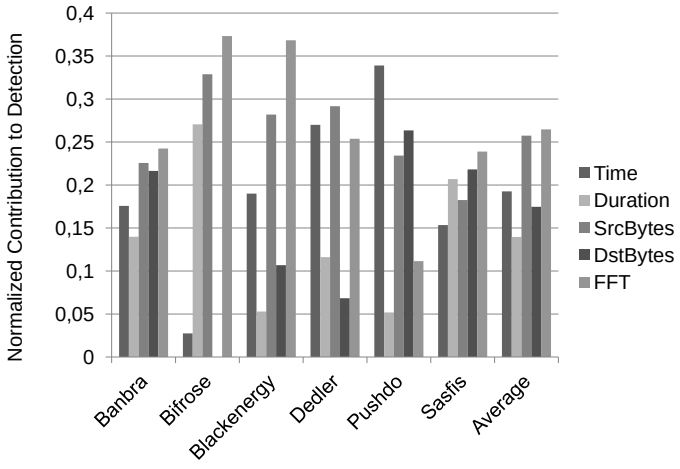


Figure 4: Normalized contribution of the different features toward a successful detection.

logarithmic scaling) whereas the detection rate decreases roughly linearly. This yields to a good threshold of $a \in [1.7, 2.0]$ — compare the lower left corner of Figure 3(b) – with good detection rates and reasonably low false positives. For an acceptance threshold of $a = 1.8$ we achieve 77% detection rate with $5 \cdot 10^{-6}$ false positives. For this parameter, Table 3 shows the detection rates of the individual malware families, averaged over the 50 cross-validation runs. Here, all Banbra samples and $\approx 85\%$ of the Blackenergy, Pushdo and Sasfis samples were detected. The only false positives were raised by Blackenergy (2) and Sasfis (1).

As one can see, detection rates vary highly for different bot families. For example, the Banbra samples all show highly periodic behavior and are very similar, which allows for good clustering quality, and, as a consequence, 100% detection rate. Blackenergy has relatively weak clusters (high standard deviation) but is still producing a roughly similar behavior. Unfortunately, the "broader" clusters lead to a higher false positive rate. Bifrose has the lowest detection rate among the malware families analyzed, which is a result of highly diverse – and therefore low quality – clusters as described in Section 4. Additionally, the quality of the Bifrose traces themselves is lower than for other bot families due to some strong, non-periodic behavior.

Another interesting experiment is the more realistic analysis of the *LabCapture* dataset in daily intervals, similar to a system administrator checking the network daily. More precisely, the traffic captures are split into separate files spanning one day each and analyzed in slices via BOTFINDER. Overall, 14 false positives – 12 Blackenergy and 2 Pushdo – were observed over the whole 2.5 month time span.

5.2.1 Contribution of Features toward Detection

To asses the quality of BOTFINDER's detection algorithm and the weighting of the different features toward a successful detection, we extracted the normalized contribution of each feature to γ_m. Figure 4 shows the averaged contribution of each feature to successful trace-to-malware identification.

Interestingly, we found fundamentally different distributions for the bots under investigation: Whereas the bot families of Banbra and Sasfis are equally periodic – and thereby well detectable – in each dimension, the remaining bots show significant discrepancies between the features. For Pushdo, the duration and the FFT is of lower significance for detection, which is primarily based on the average time interval and the number of bytes transmitted on average. The feature of destination bytes is of low importance for the remaining three bot families Bifrose, Blackenergy and Dedler, whereby Bifrose does not benefit from the feature at all. However, the source byte destination – the request toward the *C&C* server – highly contributes toward detection.

Of special interest is the impact of the Fast Fourier Transform, especially considering that the FFT accounts for the vast majority of the overall computational complexity of BOTFINDER. For all malware families except Dedler and Pushdo, the FFT is the most significant feature to detect a malware infection in the network traffic. Hereby, Bifrose is of special interest, as the average time feature contributes only minimally toward detection whereas the FFT contributes most. This indicates a much better quality and periodicity of underlying frequencies compared to the simple averaging – an indication that is verified under inspection of the underlying models which cluster significantly better for the FFT frequencies.

For the averaged contribution over all malware families (the rightmost bars in Figure 4), a relatively balanced contribution is observed. Still, the FFT and the source bytes dimension contribute slightly more toward successful detection than, e.g., the average duration or the destination bytes. Considering the mode of operation of typical bots, this outcome fits the concept of bots sending similar requests to the *C&C* and receive answers of changing size. Additionally, the superiority of underlying FFT frequencies over simple averages for the flow interval times can be seen.

Table 3: Detection rate and false positive results of BOTFINDER (acceptance threshold a=1.8) in the cross-validation experiment and compared to *BotHunter*.

Malware Family	BOTFINDER Detection	BOTFINDER False Positives	*BotHunter* Detection
Banbra	100%	0	24%
Bifrose	49%	0	0%
Blackenergy	85%	2	21%
Dedler	63%	0	n/a
Pushdo	81%	0	11%
Sasfis	87%	1	0%

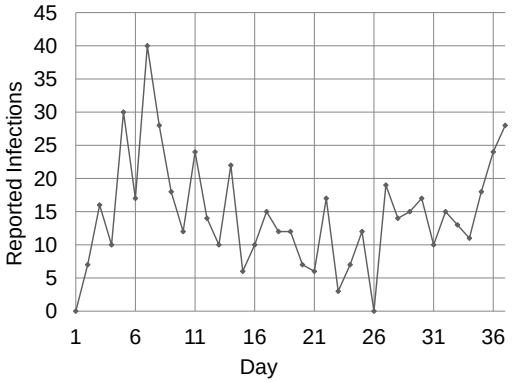


Figure 5: On average, BOTFINDER reported 15 infections per day.

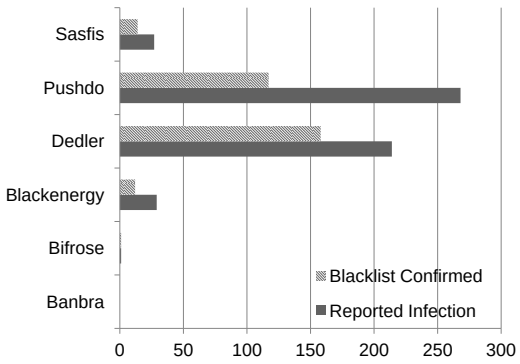


Figure 6: Reported infections by malware family. 56% of the reported malware traces had a destination IP to a know malicious, blacklisted host.

5.3 Comparison with BotHunter

Although BOTFINDER is content-agnostic, we compare our system to the well-known packet inspecting bot detection system *BotHunter* since – to the best of our knowledge – no other system allows individual bot detection in a content-agnostic manner. *BotHunter* [17] is a sophisticated bot detection system that relies on a substantially-modified Snort[5] intrusion detection system for flow identification combined with anomaly detection. It leverages detection mechanisms on the whole infection and malware execution lifecycle: Port scanning activities and dangerous binary transfers (e.g., encoded or encrypted HTTP POSTs or shell code) are used to detect the first step of the infection process. Malware downloads ("egg downloads") and, eventually, structural information regarding the command and control server plus IP blacklisting of multiple list providers are used to identify infected hosts. The later released *BotMiner* [16] adds horizontal correlation between multiple hosts, which is not in scope of this paper. Furthermore, *BotHunter* is made publicly available[6] by the authors.

We ran version 1.6.0 of *BotHunter* on full traffic dumps of our training samples and the *LabCapture* dataset. We installed the system strictly following the User Guide[7] and configured it for batch processing. We chose this *BotHunter* version as its release time fits the time of execution of our *LabCapture* traffic capture and approximates the execution time of our malware samples. As can be seen in Table 3, very few alarms were raised by *BotHunter* for the training samples. The detection rate varies between 0 and 24 percent for the different families, and we observed a high dependency on IP blacklisting for successful detection. Note that *BotHunter* had access to the full payload for the experiments, while BOTFINDER only operates on flows.

The low detection rates of *BotHunter* are – to the best of our understanding – a result of the different detection approaches and the weighting that *BotHunter* assigns to its various detection steps. *BotHunter* uses an algorithm that classifies network communication events as potential dialog steps in a malware infection process. It then correlates the observed network dialog events and compares the result to an abstract malware infection lifecycle model. Hereby, *BotHunter* works on a large number of dialog classes such as scanning, inbound attacks like web hijack attempts, DNS lookups of the client to known *C&C* sites, egg downloads, connections to a monitored Russian Business Network site, outbound attacks and scans, P2P coordination, and outbound connections to known malware sites. As we injected the malware communication into the *LabCapture* dataset, our experimental setup does not reproduce the complete malware infection lifecycle. In particular, the victim host infection and the binary download was not observable by *BotHunter*. However, these steps seem to significantly contribute to raise the score above *BotHunter*'s predefined alarm threshold (and, since they are missing, cause *BotHunter* to miss the malicious traffic). This finding is consistent with the following analysis on the mixed dataset.

Regarding the test with *BotHunter* on the malware traces mixed with the *LabCapture* dataset, we received alarms for 41 distinct IP addresses. For four IP addresses with a significant peak of alarms, we could actually confirm a bot infection, as researchers executed malware in a bridged VM. As BOTFINDER was not trained for the specific bot family that the researcher was working on, it is not surprising that BOTFINDER missed this infection. For most IP addresses (37), we were unable to confirm an actual infection. Often, the connections were made to IRC servers (BitCoin trades) or raised because of the high NXDOMAIN activity of the University's core router. In another instance, *BotHunter* identified the download of an Ubuntu Natty ISO as an exploit (Windows Packed Executable and egg download). This shows that the number of false positives is significantly higher than those raised by our system on the same traffic.

5.4 ISPNetFlow Analysis

The *ISPNetFlow* dataset is the most challenging dataset to analyze, as we do not have much information about the associated network. We trained BOTFINDER with all available training malware traces and applied BOTFINDER to the dataset.

Overall, BOTFINDER labeled 542 traces as evidence of bot infections, which corresponds to an average of 14.6 alerts per day. This number of events can be easily handled by a system administrator, manually during daily operations or by triggering an automated user notification about potential problems. Figure 5 shows the evolution of infections over the analysis time frame, which varies from days with no infections at all to days with a maximum of 40 reported infections. Figure 6 illustrates the total number of reported incidents per bot. Pushdo and Dedler are dominating the detected infections with 268 and 214 reports, respectively, followed by Sasfis with 14 and Blackenergy with 12. Bifrose was found only once in the traffic.

We investigated the IP addresses involved in suspicious connections to judge the detection quality of BOTFINDER. We received the internal IP ranges from the *ISPNetFlow* system administrators and were able to split the set of all involved IPs into internal and external IP addresses. Only two out of the 542 traces had their source and destination IP addresses both inside the network. This indicates that our system is not generating many – most probably false – indications of internal infections where both the bot and the *C&C* server are inside the observed network. We compared the remaining 540 external IP addresses to a number of publicly available blacklists[8] and *had a positive match for 302 IPs or 56%*. This result strongly supports our hypothesis that BOTFINDER is able to identify real malware infections with a relatively low number of false positives.

Whereas the 302 blacklist-confirmed IP addresses do not strongly

[5] `http://www.snort.org`
[6] `http://www.bothunter.org/`
[7] `http://www.bothunter.net/OnlinePDF.html`

[8] The RBLS `http://rbls.org/` service allows to analyze a large number of blacklists using a single query. We ignored "RFC-ignorant" listings.

Table 4: Top-5 aggregated clusters of non-blacklisted destination IP addresses in the *ISPNetFlow* dataset.

Size	Service or Organization
46	Apple Inc.
21	A company that offers web services and dedicated servers
7	A Russian BitTorrent tracker
6	A company that offers dedicated servers
5	NTT / Akamai

cluster to specific networks, the 238 non-confirmed IP addresses show multiple large clusters and in total, 85 IPs contribute to the Top-5 destination networks. Table 4 lists the companies or services offered by the Top-5 found in the list of not blacklisted destination IP addresses. Overall, we found 46 destination IP addresses to point to servers of the Apple Incorporation, which can be considered a false positive. Two services offer a variety of web services and advertise dedicated web servers for rent. Although only speculation, a possible cause might be malware authors that rent dedicated servers. Paying by maliciously obtained payment information allows botmasters access to *C&C* servers while hiding their trails and evading law enforcement. However, this assumption is unable to be verified in the scope of this paper. A company located in the Seychelles offers a Russian BitTorrent tracker. No information is available on the service offered at the final 5 destination IPs which point to NTT / Akamai.

If we add Apple Inc. to the whitelist, effectively a rate of 61% (302 of 496 destination IP addresses) matching blacklist-entries is observed. Considering that various not blacklisted destination IP addresses belong to rented dedicated servers or other web providers, it is a reasonable assumption that a significant fraction of the 194 not blacklisted IP addresses actually belong to malicious servers.

5.5 Summary

In our evaluation we showed that BOTFINDER

- has a detection rate around 80% for a parameter setting with low false positives,
- outperforms the content-inspection-based IDS *BotHunter*,
- is able to operate on large datasets,
- and identifies likely true positive connections to *C&C* hosts in the *ISPNetFlow* dataset.

Overall, BOTFINDER proved that the malware families under investigation exhibit a behavior regular enough to allow high detection rates using network-statistics only.

6. BOT EVOLUTION

By detecting malware without relying on deep packet (content) inspection – which is an inherently difficult task – BOTFINDER raises the bar for malware authors and might trigger a new round of bot evolution. In the following, we will introduce potential evasion techniques that malware authors might try to thwart BOTFINDER, and we discuss how we can handle these techniques.

6.1 Adding Randomness

We assume regularity in the communication between bots and their *C&C* servers and showed that this assumption holds for the bots under investigation. Nevertheless, malware authors might intentionally modify the communication patterns of their bots to evade detection, as suggested, for example, by Stinson et al. [31]. More specifically, botnet authors could randomize the time between connections from the bot to the *C&C* server or the number of bytes that are exchanged. For the botmaster, this comes at the price of loss of network agility and degraded information propagation within the botnet. However, by using randomization techniques, the malware author effectively decreases the quality of the trace for BOTFINDER, which lowers the detection quality. Interestingly, BOTFINDER already operates on highly fluctuating traces and is, as our detection results show, robust against significant randomization around the average. To further illustrate BOTFINDER's resilience against randomization, we analyzed the *C&C* trace detection rate with (artificially) increasing randomization. A randomization of 50% means that we subtract or add up to 50% of the mean value. For example, for an interval of 100 seconds and a randomization rate of 20%, we obtain a new interval between 80 and 120 seconds. Figure 7(a) shows the effect on the detection rate of BOTFINDER with randomization on the time (impacting the "average time" and the "FFT" feature), randomization of time and "duration," and with randomized "source bytes", "destination bytes," and "duration". As can be seen, BOTFINDER's detection rate drops slightly but remains stable above 60% even when the randomization reaches 100%.

6.2 Introducing Larger Gaps

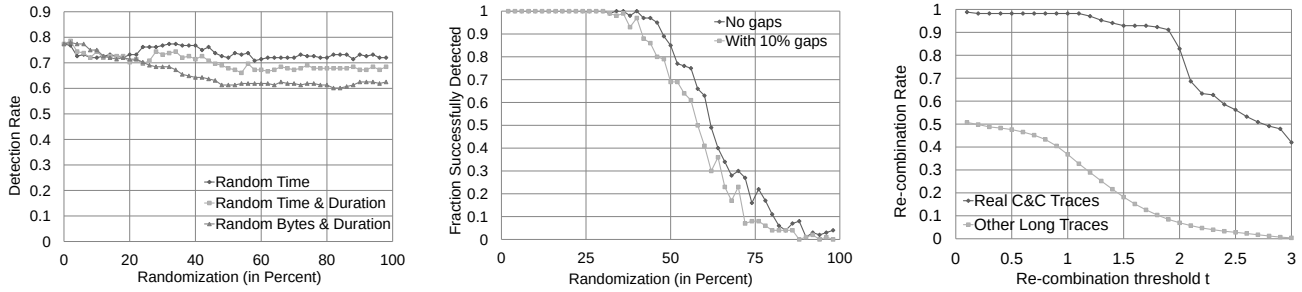
Malware authors might try to evade detection by adding longer intervals of inactivity between *C&C* connections. In this case, the Fast Fourier Transformation significantly increases BOTFINDER's detection capabilities: Due to its ability to separate different *C&C* communication periodicities, the introduction of large gaps into the trace (which impacts the average) does not significantly reduce the FFT detection rate. For a randomization between 0 and 100 percent of the base frequency, Figure 7(b) shows the fraction of FFTs that detected the correct, underlying communication frequency. As can be seen, the introduction of large, randomly distributed long gaps does not significantly reduce the detection quality of the FFT-based models.

6.3 High Fluctuation of C&C Servers

Malware programs might try to exploit the fact that BOTFINDER requires collection of a certain, minimal amount of data for analysis. Now, if the *C&C* server IP addresses are changed very frequently, BOTFINDER cannot build traces of minimal length $|\mathcal{T}|_{min} = 50$. *Currently, we do not observe such high C&C server fluctuations (IP flux) in our collected malware data.* Even a highly domain-fluxing malware, such as Torpig (as analyzed in [32]) uses two main communication intervals of 20 minutes (for upload of stolen data) and 2 hours for updating server information. Still, Torpig changes the *C&C* server domain in weekly intervals. Nevertheless, we already introduce a countermeasure (that might also help with P2P botnets) by using elements of horizontal correlation. This is an additional pre-processing step that operates before the full feature extraction (Step 4 in Figure 1). The step constructs longer traces from shorter traces (e.g, of length 20 to 49) that exhibit similar statistical features. Hereby, we again utilize the observation that *C&C* communication exhibits higher regularity than other frequent communication.

To decide whether to merge two sub-traces $\mathcal{T}_A$ and $\mathcal{T}_B$, we use two factors:

- We require that the standard deviation of the combined $\mathcal{T}_{AB}$ is lower than the standard deviation of at least one of the individual traces. Thereby, traces around a significantly different average – even with relatively low fluctuations – do not match and are automatically excluded.
- We use a quality rating function analog to the model-matching algorithm to rate each individual feature. If the sum over all



(a) Randomization Impact (b) FFT Detection (c) Pre-Processing

Figure 7: (a) depicts BOTFINDER's detection rate with increasing randomization, (b) summarizes the degrading of the FFT periodicity detection capability, and (c) depicts the re-combination ratio during the pre-processing step of real *C&C* and other, long traces.

feature-qualities of the combined trace $\mathcal{T}_{AB}$ is above a threshold t, we accept the trace recombination.

The lower the value of t is, the more sub-trace combinations we accept and the higher the additional workload for BOTFINDER is. We applied the presented pre-processing step on real-world data and investigated:

- The ability to re-assemble real *C&C* traces.
- The re-assembly difference between real *C&C* and other, long traces.
- The amount of additional traces that need to be analyzed by BOTFINDER and the implied additional workload.
- The false positive ratio of the newly generated traces.

The impact of the higher periodicity of *C&C* traffic can be clearly seen in Figure 7(c), which illustrates the re-assembly rates of bisected real *C&C* traces and of long non-*C&C* traces for different acceptance thresholds t. For $t = 1.9$, BOTFINDER *re-assembled 91% of the real C&C traces* and combined only 8% of non-*C&C* long traces.

Further, when running on the 2.5 months of *LabCapture* data, we (incorrectly) re-assembled only 3.4 million new traces. Using the same detection threshold as in our evaluation ($a = 1.8$), this does not introduce any new false positives. As we would typically run pre-processing over shorter time frames – as it is a countermeasure against fast flux – even fewer new traces will be generated. For example, in a ten day NetFlow traffic set, only 0.6% of the IP addresses with more than one sub-trace generated additional traces. Computing these traces increased the workload for BOTFINDER by 85% compared to normal operation. That is, with a modest increase in overhead, BOTFINDER also covers cases where bots frequently switch IP addresses.

6.4 P2P Bots

BOTFINDER might be able to detect P2P networks by concatenating the communication to different peers in one trace. Nevertheless, complementing BOTFINDER with elements from different existing different approaches might be beneficial: BOTFINDER could be expanded by a component that creates structural behavior graphs, as proposed by Gu et al. [16, 17], or be complemented by P2P net analysis techniques similar to BotTrack [12] or BotGrep [24], which try to reveal members of a bot network by surveillance of a single member of the network. Still, completely changing to a P2P-based botnet also imposes significant challenges for the botmaster. These include the ease of enumeration of all participating bots by every member in the botnet (for example, a honeypot-caught bot under control of a security researcher as performed by BotGrep, and the time to disseminate commands. Hence, most botnets today use a centralized infrastructure.

6.5 Bot-like Benign Traffic

Although unlikely, benign communication might accidentally exhibit a similar traffic pattern as a bot family. For example, a POP3 mail server might get queried in the same interval as a bot communicates with its *C&C* server, and the traffic sizes might accidentally match. If these services operate on a static IP, a system administrator can easily exclude these false positives by whitelisting this IP address. A local BOTFINDER installation should be configured to ignore communication between hosts under the same local authority. For popular web services with similar features, a generic whitelisting is possible.

6.6 Discussion

BOTFINDER is able to learn new communication patterns during training and is robust against the addition of randomized traffic or large gaps. Furthermore, given the pre-processing step, even changing the *C&C* server frequently is highly likely to be detected. Nevertheless, BOTFINDER is completely reliant on statistical data and regularities. If the attacker is willing to:

- significantly randomize the bot's communication pattern,
- drastically increase the communication intervals to force BOTFINDER to capture traces over longer periods of time,
- introduce overhead traffic for source and destination byte variation,
- change the *C&C* server extremely frequently, e.g., after each tenth communication,
- use completely different traffic patterns after each *C&C* server change, then

BOTFINDER's detection fails as minimal or no statistical consistency can be found anymore. On the contrary, a malware author who implements such evasion techniques, has to trade the botnets performance in order to evade BOTFINDER: Using randomization and additional traffic increases the overhead and reduces synchronization and the network-agility of the botnet. In particular, especially the frequent change of *C&C* servers is costly and requires an increased amount of work and cost by the botmaster: Domains

need to be pre-registered and paid and new globally routeable IP addresses must be obtained. Hereby, the bots need to know to which *C&C* server to connect, so the new domains must either follow a pre-defined and malware-hardcoded pattern – which allows take-over attacks by security researchers such as in Stone-Gross et al. [32] (with a weekly changing domain) – or lists of new *C&C* servers need to be distributed to the members of the botnet. Both ways increase the botnet operator's costs and reduce stability and performance of the malware network.

7. RELATED WORK

Research in network based bot detection can be classified into two main directions: In *vertical correlation* the network traffic is inspected for evidence of bot infections, such as scanning, spamming or command and control communication. *BotHunter* [17], for instance, applies a combination of signature and anomaly-based intrusion detection components to detect a typical infection lifecycle, whereas Rishi [15] and Binkley et al. [3] examine and model IRC-based network traffic for nickname patterns that are frequently used by bots. Unfortunately, some of these techniques are tailored to a specific botnet structure [3, 15], or rely on the presence of a specific bot-infection lifecycle [17]. Moreover, most techniques rely on the presence of noisy behavior such as scans, spam, or DoS attack traffic. Wurzinger et al. [36] and Perdisci et al. [27] automatically generate signatures that represent the behavior of an infected host. The key point in these strategies is that bots receive commands from the bot master and then respond in specific ways that allow signature generation.

Although these approaches are shown to achieve a very high detection rate and a limited false positives ratio, they require inspecting packet content and can thus be circumvented by encrypting the *C&C* communication. Giroire et al. [14] presented an approach which is similar to BOTFINDER in this aspect as both focus on temporal relationships. However, our system differs fundamentally in the way malware detection is performed. In particular, Giroire's work is based on the concept of destination atoms and persistence. Destination atoms group together communication toward a common service or web-address, whereas the persistence is a multi-granular measure of destination atoms' temporal regularity. The main idea is to observe the per-host initiated connections for a certain (training) period and group them into destination atoms. Subsequently, very persistent destination atoms form a host's whitelist, which will be compared against the very persistent destination atoms found once the training session ends. Thus, very persistent destination atoms will be flagged as anomalous and potentially identify a *C&C* host.

The second direction is the *horizontal correlation* of network events from two or more hosts, which are involved in similar, malicious communication. Interesting approaches are represented by BotSniffer [18], BotMiner [16], TAMD [37], and the work by Strayer et al. [33]. Except the latter, which works on IRC analysis, the main strength of these systems is their independence of the underlying botnet structure, and thus, they have shown to be effective in detecting pull-, push-, and P2P-based botnets. By contrast, 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 single bot-infected hosts. In addition, the detection mechanisms require that some noisy behavior is observed [16]. Unfortunately, low-pace, non-noisy, and profit-driven behavior [11, 19] is increasing in bots as witnessed in the past few years [32].

Another way to detect P2P botnets is shown by BotGrep [24], BotTrack [12], and Coskun et al. [7] "Friends of An Enemy" (FoE), which leverage the underlying communication infrastructure in the P2P overlay. Whereas BotGrep uses specifics of the DHT interactions, BotTrack operates on NetFlows only and is comparable to BOTFINDER in this aspect. For FoE [7], mutual communication graphs are calculated based on mutual communication without packet content inspection. However, all systems need to be bootstrapped with the botnet under investigation, typically by utilizing a participating active malware sample in a honeypot. Connections of this bot under surveillance reveal other members of the network. This requirement of an active source in the honeypot is a significant drawback. Nevertheless, concepts from these solutions might complement BOTFINDER to allow detection of P2P based bots during NetFlow analysis as well.

8. FUTURE WORK

BOTFINDER can be seen as a bot detection framework that allows improvement on multiple layers. Potential future optimizations, for example, cover:

1: The sandboxed training environment can be improved to better circumvent malware authors to probe for virtual machine settings and react by stopping all activity.

2: A separate problem – of the general anti malware research community – is the classification of malware to families. Currently we rely on Anubis signatures and VirusTotal labels which yield sufficiently good results – especially because we drop small clusters in the model building process and thereby rely only on more persistent features among multiple binary malware samples. However, more accurate classifiers will definitely benefit our system.

3: We would also like to experiment with unsupervised learning approaches in the training phase. Hereby, a machine learning algorithm might be able to select the ideal features that describe a given malware family best and weight the features correspondingly for the detection phase.

4: The malware detection might be improved by more sophisticated features that do not exploit periodicity alone but periodicity of communication sequences learned in the training phase, such as recurring three times a 20 minutes interval followed by a longer gap of 2 hours.

A substantial advantage is the fact that many improvements can be performed on the training side alone, which makes changes to any deployed BOTFINDER scanning installation unnecessary.

9. CONCLUSION

We presented BOTFINDER, a novel malware detection system that is based on statistical network flow analysis. After a fully-automated, non-supervised training phase on known bot families, BOTFINDER creates models based on the clustered statistical features of command and control communication. We showed that the general assumption of *C&C* communication being periodic holds and that BOTFINDER is able to detect malware infections with high detection rates of nearly 80% via traffic pattern analysis. A significant novelty is BOTFINDER's ability to detect malware without the need of IP blacklisting or deep packet inspection. BOTFINDER is therefore able to operate on fully-encrypted traffic, raising the bar for malware authors.

10. ACKNOWLEDGEMENTS

We gracefully acknowledge the help of Bryce Boe. This work was supported by the Office of Naval Research (ONR) under Grant N000140911042, the Army Research Office (ARO) under grant W911NF0910553, and the NSF under grants CNS-0845559 and CNS-0905537, as well as Secure Business Austria.

11. REFERENCES

[1] U. Bayer, P. M. Comparetti, C. Hlauschek, C. Kruegel, and E. Kirda. Scalable, Behavior-Based Malware Clustering. In *NDSS*, 2009.

[2] U. Bayer, C. Kruegel, and E. Kirda. Anubis: Analyzing Unknown Binaries. In *http://anubis.iseclab.org/*, 2008.

[3] J. R. Binkley. An algorithm for anomaly-based botnet detection. In *SRUTI*, 2006.

[4] F. Chang, W. Qiu, R. H. Zamar, R. Lazarus, and X. Wang. clues: An R Package for Nonparametric Clustering Based on Local Shrinking. *Journal of Statistical Software*, 33(4):1–16, 2 2010.

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

[6] E. Cooke, F. Jahanian, and D. McPherson. The Zombie roundup: understanding, detecting, and disrupting botnets. In *SRUTI*, 2005.

[7] B. Coskun, S. Dietrich, and N. Memon. Friends of An Enemy: Identifying Local Members of Peer-to-Peer Botnets Using Mutual Contacts. In *ACSAC*, 2010.

[8] A. Dinaburg, P. Royal, M. Sharif, and W. Lee. Ether: malware analysis via hardware virtualization extensions. In *ACM CCS*, 2008.

[9] P. Fogla and W. Lee. Evading network anomaly detection systems: formal reasoning and practical techniques. In *ACM CCS*, 2006.

[10] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *USENIX Security*, 2006.

[11] J. Franklin, V. Paxson, A. Perrig, and S. Savage. An Inquiry into the Nature and Causes of the Wealth of Internet Miscreants. In *ACM CCS*, 2007.

[12] J. Franï£¡ois, S. Wang, R. State, and T. Engel. Bottrack: Tracking botnets using netflow and pagerank. In *IFIP Networking*. 2011.

[13] F. Freiling, T. Holz, and G. Wicherski. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In *ESORICS*, 2005.

[14] F. Giroire, J. Chandrashekar, N. Taft, E. M. Schooler, and D. Papagiannaki. Exploiting Temporal Persistence to Detect Covert Botnet Channels. In *RAID*, 2009.

[15] J. Goebel and T. Holz. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. In *USENIX HotBots*, 2007.

[16] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In *USENIX Security*, 2008.

[17] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In *USENIX Security*, 2007.

[18] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *NDSS*, 2008.

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

[20] J. A. Hartigan and M. A. Wong. A k-means clustering algorithm. *JSTOR: Applied Statistics*, 28(1), 1979.

[21] G. Jacob, R. Hund, C. Kruegel, and T. Holz. Jackstraws: Picking Command and Control Connections from Bot Traffic. *USENIX Security*, 2011.

[22] L. Kaufman and P. Rousseeuw. *Finding Groups in Data An Introduction to Cluster Analysis*. Wiley Interscience, New York, 1990.

[23] S. Kundu. Gravitational clustering: a new approach based on the spatial distribution of the points. *Pattern Recognition*, 32(7):1149 – 1160, 1999.

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

[25] V. Paxson. Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[26] R. Perdisci, D. Dagon, P. Fogla, and M. Sharif. Misleading worm signature generators using deliberate noise injection. In *IEEE S&P*, 2006.

[27] R. Perdisci, W. Lee, and N. Feamster. Behavioral clustering of http-based malware and signature generation using malicious network traces. In *USENIX NSDI*, 2010.

[28] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2010.

[29] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *ACM IMC*, 2006.

[30] D. Song, D. Brumley, H. Yin, J. Caballero, I. Jager, M. Kang, Z. Liang, J. Newsome, P. Poosankam, and P. Saxena. BitBlaze: A New Approach to Computer Security via Binary Analysis. In *ICISS*. 2008.

[31] E. Stinson and J. C. Mitchell. Towards systematic evaluation of the evadability of bot/botnet detection methods. In *USENIX WOOT*, 2008.

[32] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your Botnet is My Botnet: Analysis of a Botnet Takeover. In *ACM CCS*, 2009.

[33] 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.

[34] X. Wang, W. Qiu, and R. H. Zamar. Clues: A non-parametric clustering method based on local shrinking. *Computational Statistics and Data Analysis*, 52(1):286 – 298, 2007.

[35] C. Willems, T. Holz, and F. Freiling. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE S&P*, 2007.

[36] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically Generating Models for Botnet Detection. In *ESORICS*, 2009.

[37] T.-F. Yen and M. K. Reiter. Traffic Aggregation for Malware Detection. In *DIMVA*, 2008.