Extracting Probable Command and Control Signatures for Detecting Botnets

Ali Zand, Giovanni Vigna, Xifeng Yan, Christopher Kruegel
University of California, Santa Barbara
{zand,vigna,xyan,chris}@cs.ucsb.edu

ABSTRACT

Botnets, which are networks of compromised machines under the control of a single malicious entity, are a serious threat to online security. The fact that botnets, by definition, receive their commands from a single entity can be leveraged to fight them. To this end, one requires techniques that can detect command and control (C&C) traffic, as well as the servers that host C&C services. Given the knowledge of a C&C server's IP address, one can use this information to detect all hosts that attempt to contact such a server, and subsequently disinfect, disable, or block the infected machines. This information can also be used by law enforcement to take down the C&C server.

In this paper, we present a new botnet C&C signature extraction approach that can be used to find C&C communication in traffic generated by executing malware samples in a dynamic analysis system. This approach works in two steps. First, we extract all frequent strings seen in the network traffic. Second, we use a function that assigns a score to each string. This score represents the likelihood that the string is indicative of C&C traffic. This function allows us to rank strings and focus our attention on those that likely represent good C&C signatures. We apply our technique to almost 2.6 million network connections produced by running more than 1.4 million malware samples. Using our technique, we were able to automatically extract a set of signatures that are able to identify C&C traffic. Furthermore, we compared our signatures with those used by existing tools, such as Snort and BotHunter.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*Invasive Software*

General Terms

Security

1. INTRODUCTION

A botnet is a collection of compromised machines controlled by a single entity (botmaster). The botmaster communicates (sends commands and receives information) with his/her bots over a command and control (C&C) channel. Nowadays, P2P, IRC, and HTTP are the most popular protocols for implementing botnet C&C channels [22]. One way to defend against botnets is to detect the presence of botnet C&C communication. When C&C traffic is detected, one can prevent commands from reaching the victims, track the C&C servers and shut them down [22], or use this information to dismantle the botnet infrastructure.

Previous work has attacked the problem of detecting bot-infected machines from different perspectives. For example, some previous systems (such as BotSniffer [11] and BotMiner [8]) attempt to detect synchronized activity coming from multiple, infected hosts in a network. Unfortunately, these approaches typically work only for large networks, because, in a small network, there are not enough infected hosts to expose synchronized or correlated activity. Another problem of these approaches is that they are able to detect the bots only after they have observed enough synchronized activity. Therefore, a bot can remain undetected for a considerable amount of time, in which it can cause damage to the infected network.

Signature-based detection systems (e.g., intrusion detection systems such as Snort, Bro, and BotHunter [9]) work independently of the monitored network size. These systems are able to detect an infected host even if they observe only a single connection matching one of their signatures. The problem with signature-based approaches is that they cannot detect new, previously-unknown, or upgraded botnets. Thus, they require frequent updates to their signature databases. Unfortunately, the current approach to developing signatures is mostly manual. This process is tedious and time-consuming.

To improve signature-based botnet detection, and to minimize the amount of manual effort required for extracting new signatures for a botnet, we propose an automated signature extraction method for detecting C&C communication. Our approach starts with a pool of network connections as input. These connections are recorded by running bots in a controlled environment (in our case, in a sandbox). The pool contains many C&C connections (belonging to many different botnets) but also many non-C&C connections that bots produce as part of their operations. Such connections are often related to malicious activites (such as scans, exploits, or spam emails) but can also contain benign requests to check for the current time or for network connectivity. Our goal is to produce strings (signatures) that characterize the C&C connections.

To produce C&C signatures, we first extract all frequent strings. Of course, frequent strings are not necessarily associated with C&C communications. Thus, we need a second step that allows us to rank (score) strings so that a string receives a high score when it likely captures C&C traffic, and a low score otherwise. The key observation underlying our ranking process is that C&C connec-

SAC'14 March 24-28, 2014, Gyeongju, Korea.

http://dx.doi.org/10.1145/2554850.2554896

tions (or, at least, the connections related to a single command) that belong to one particular botnet are relatively similar to each other. By similar, we mean that they exhibit similar network-level properties. That is, the duration of these connections is similar, they contain a similar number of packets of similar sizes exchanged between bot and server, et cetera. The reason is that a C&C protocol typically implements a simple data exchange in which the client (bot) reports a fixed number of data items and/or the server sends a command with a fixed structure. Although the content of the connections might vary quite a bit, the more general network-level properties are similar. The variance of non-C&C traffic, on the other hand, is much larger.

Our key observation (namely, C&C connections for a specific botnet are similar) can be exploited to identify likely C&C-related strings. To this end, we first group (cluster) all connections into classes that share similar network features. Then, we look for frequent strings that appear only in few classes (or, optimally, in a single class). These strings are highly indicative of the connections in this class. Since C&C connections of a botnet tend to cluster well, strings that provide significant information about an individual cluster are likely good C&C signatures. Non-C&C traffic, on the other hand, is typically spread over many classes. As a result, frequent strings in such traffic (many groups) are ranked low.

There is a subtle but important difference between our work and previous signature extraction systems (from EarlyBird [21] to Polygraph [19] and Lisabeth [4]). The difference is that previous systems either operate on a malicious pool that contains only (or, at least, almost exclusively) known, malicious connections, or they assume that there is a way to distinguish between a suspicious/malicious connection and a legitimate one. As a result, these systems try to find the smallest number of strings that cover most elements in the malicious pool without matching connections in a second pool that holds benign traffic. In our problem setting, we do not assume such access to a pool that contains mostly C&C connections. In fact, the key problem for us is to distinguish C&C from non-C&C connections, even though both types of traffic are produced by the same set of malware samples.

To summarize, the main contributions of this paper are the following:

- We present a novel approach for automatically extracting string signatures that capture botnet C&C messages.
- We apply different techniques to determine the quality of extracted signatures, based on the amount of information they provide about the traffic.
- We apply our technique to almost 2.6 million network connections produced by more than 1.4 million malware samples running in a dynamic malware analysis system. Furthermore, we evaluate the quality of the resulting candidate signatures, and we found that our system generates signature candidates with high accuracy.

2. FINDING C&C SIGNATURES

The signature mining process involves three main phases: extracting frequent strings, detecting distinguishing strings, and an optional phase of merging candidate strings. This process is shown in Figure 1.

We start with a pool of network connections recorded by executing malware in a dynamic analysis system. First, the contents of these connections are examined to extract the frequent strings. These strings are extracted using a generalized suffix tree [24]. In the second phase, we extract the statistical features of the connections and use these features to cluster the connections into classes of similar traffic. The connection clusters, based on the traffic's statistical features, are then used to select the distinguishing strings among the frequent strings. Finally, in the third (and optional) phase, candidate strings are merged to produce more precise signatures. The details of each phase are discussed in the following sections.

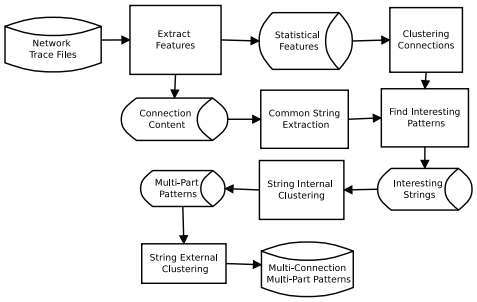

Figure 1: Process Flowchart

2.1 Frequent String Extraction

In this phase, the most frequent substrings are extracted from the traffic connections. We extract all substrings that appear at least n times with length greater than or equal to k characters. We experimented with different values for n and k, and we selected $n = 10$ and $k = 5$. These were the the smallest values that resulted in a set of meaningful candidate signatures. In general, selecting a larger n makes string ranking more accurate, at the cost of losing some candidate strings. On the other hand, if n is too small, many random combinations of characters will appear in the result set, and it would be hard to claim anything about their value as a signature (because they appeared in a very small number of cases only). A larger value for k results in more accurate signatures, while it prevents smaller strings from appearing in the result set. On the other hand, a small k value results in random strings of characters to be returned as frequent strings. To extract the strings, we use a generalized suffix tree [24].

At the end of this step, we have a set of frequent strings. This set will likely contain valid C&C signatures, but also irrelevant strings that are not related to C&C communication. We attempt to eliminate the irrelevant strings in the next step.

2.2 Ranking Frequent Strings

In this phase, we use a string-ranking mechanism to remove the non-distinguishing strings from the result set. A distinguishing string is a string that appears frequently in one class of connections (the traffic generated by the C&C mechanism of one malware family) and infrequently in other ones [12]. Non-distinguishing strings cannot be good signatures, because they appear in a number of different classes of traffic, and hence, they cannot identify a specific C&C protocol.

Not all frequent strings are good signature candidates. For example, many of the frequent patterns are indicative of the transport or application protocol (strings such as "HTTP," "GET," "POST," etc.). These strings appear in a wide range of applications that use the protocol. The goal of the ranking method is to rank the strings so that more distinguishing strings are placed higher in the ranking. We use the information gain of different frequent strings for ranking them.

2.2.1 Traffic Clustering

As mentioned previously, as a prerequisite for the information theoretic ranking, it is necessary to cluster all network connections

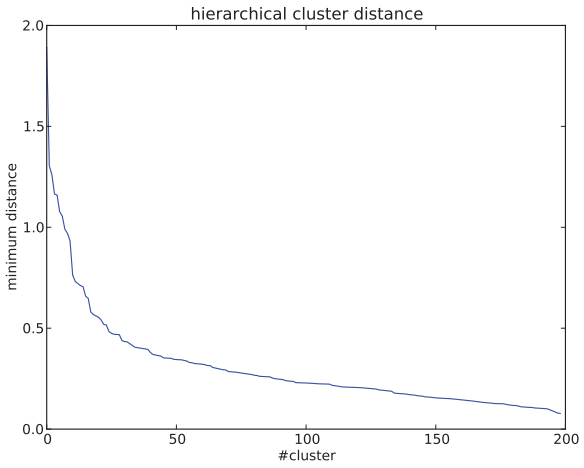

Figure 2: Hierarchical clustering minimum distance

first. To this end, we use the following statistical features extracted from each network connection:

- connection duration in milliseconds
- number of bytes sent by client/server (2 attributes)
- variance of data size sent by client/server (2 attributes)
- average/variance of inter-arrival time of the packets (2 attributes)
- number of packets with specific size ranges [0-99, 100-199, 200-299, 300-399, 400-499, >=500] (6 attributes)
- whether the connection is initiated from/to the remote site (2 attributes)
- number of packets sent by client/server (2 attributes)

We use the K-means clustering algorithm with the Euclidean distance function on the normalized values of the features during the clustering process.

To find an appropriate K (i.e., the optimal number of clusters to be used by the K-means algorithm), we iterated over increasing values for K while observing the resulting minimum distance between clusters. As shown in Figure 2, we have a steep decrease in the minimum distance of the clusters as we increase the number of clusters until we reach 50 clusters. At this point, increasing K no longer yields significantly smaller clusters distances.

2.2.2 Information Theoretic Ranking Criterion

From an information theoretic point of view, a good signature is one with the higher information gain. A similar approach is used in machine learning for selecting features for building decision trees [18]. If X and Y are two random quantities, and X is the quantity to learn about (e.g., the class of traffic), and Y is the feature used to determine X (e.g., the features), the information gain of feature Y is computed by $I(X;Y) = H(X) - H(X|Y)$, in which $H(X|Y)$ is the conditional entropy of X when Y is known. $I(X;Y)$ is sometimes called the average mutual information between X and Y [7]. A feature with a higher information gain is a better feature to learn the target function.

We define the normalized information gain (I_n) as $I_n(X;Y) = \frac{I(X;Y)}{H(X)} = 1 - \frac{H(X|Y)}{H(X)}$. We call $E_n = \frac{H(X|Y)}{H(X)}$ the normalized entropy, and minimizing it is equivalent to maximizing the information gain.

To compute the normalized entropy of a string, we count the number of its instances in each cluster. If a string does not provide any information about the clusters, we expect it to appear in each of them uniformly. In other words, $H(X)$, which is the entropy without any knowledge of Y, will be the entropy of the uniform distribution of the string among all clusters. On the other hand, $H(X|Y)$ is the entropy of the actual distribution of the string among the clusters. We compute the normalized entropy using these values.

The actual entropy can be expressed as $Ent(dist) = \sum_{k=1}^{n} P_k * ln(P_k)$, where n is the number of clusters and P_k is the probability of appearance of a given string in cluster k. P_k is computed by taking the number of appearances of the given string in cluster k divided by the total number of its appearances. The normalized entropy E_n is computed using the following formula: $E_n = \frac{Ent}{Ent_{max}}$. $Ent_{max} = Ent(uniform\_distribution)$, where $Ent(uniform\_distribution)$ is the entropy of the given string being distributed as uniformly as possible among clusters (all clusters having the same number of strings or at most a difference of 1).

For example, assume we have four clusters, and the string s appears four times, namely two times in cluster 1, and two times in cluster 2. Let us consider the uniform distribution of s among clusters is the case in which s appears once in each cluster:
$H(X) = \frac{1}{4} \times \log(\frac{1}{\frac{1}{4}}) + \frac{1}{4} \times \log(\frac{1}{\frac{1}{4}}) + \frac{1}{4} \times \log(\frac{1}{\frac{1}{4}}) + \frac{1}{4} \times \log(\frac{1}{\frac{1}{4}}) = 2$
$H(X|Y) = \frac{1}{2} \times \log(\frac{1}{\frac{1}{2}}) + \frac{1}{2} \times \log(\frac{1}{\frac{1}{2}}) + 0 + 0 = 1$
$E_n = \frac{H(X|Y)}{H(X)} = \frac{1}{2}$.

Intuitively, strings that are indicative of a certain C&C protocol will appear in the same cluster, or a few number of clusters, because the corresponding connections have the same traffic statistics. Patterns that appear in many clusters are typically associated with transport or general application protocols (such as HTTP).

2.3 Combining Patterns

In the third and optional phase, strings that appear in the same set of connections can be combined to obtain longer and more precise signature patterns. That is, smaller string fragments that are part of the same C&C connection can be combined into longer, more precise signatures. For combining the extracted strings, we used a document clustering algorithm, called suffix tree algorithm, introduced by Zamir et al. [26]. This algorithm creates clusters based on the phrases shared among the documents. In this document clustering algorithm, each key term (a signature in our case) is represented as a node in a graph. An edge is introduced between two nodes (terms) when there is a relationship between these terms (for example, if the number of documents they both appear in is greater than some predetermined percentage of the number of documents each of them appears in). The nodes corresponding to s_1 and s_2 will be connected to each other if $\frac{|D_{s_1} \cap D_{s_2}|}{|D_{s_1}|} \geq c$ and $\frac{|D_{s_1} \cap D_{s_2}|}{|D_{s_2}|} \geq c$ where D_s is the set of documents in which phrase s appears, and c is a predetermined constant. Such relationships express situations where words frequently appear together in documents, or when words appear in similar documents. After building the graph, the algorithm considers each connected component a cluster.

A similar approach can be used to cluster candidate signatures. We use similar notions of relationships between strings. We adopted the suffix tree clustering algorithm in three different versions: (1) Two strings are connected when they frequently appear together in the *same* connection. (2) Alternatively, two strings are connected if they frequently appear in connections to the same destinations (based on remote IP address, protocol, and remote port number). (3) Two strings are connected when they appear in many pairs of

different connections, where each pair is produced by one malware program. An example for the last case can be a pattern for a bot where one strings appears in the bot's HTTP connection, followed by a pattern in that bot's subsequent SMTP traffic.

3. EXPERIMENTS

We applied our approach to a set of network trace files generated by a dynamic malware analysis system.

3.1 Dataset

We received the network dump data for binary files from Anubis [16], a popular dynamic malware analysis tool that offers a public submission interface. Anubis is frequently used by researchers and security companies, and it processes thousands of currently active malware samples every day. Anubis runs each program in a sandbox environment, recording the program's interactions with the operating system. It also stores the network traffic generated by each sample in a *tcpdump*. We used these *tcpdump* files as the only input to our system.

To avoid infecting publicly accessible machines on the Internet, Anubis relies on a containment policy that blocks a sample from accessing services on the Internet that are frequently the target of attacks but that are typically not used for command and control (examples include SMTP, IMAP, and SMB).

It should be noted that Anubis runs each binary (using md5 hash value of the binary as the identifier of the binary) only once and only for five minutes. Therefore, if samples of a certain malware family are infrequent in the provided data set, we may not be able to detect their C&C protocol.

For our experiments, we used 1,427,798 *tcpdump* files present in a snapshot of the system database (with an overall data volume of 260GB). After filtering out non TCP/UDP traffic, traffic with no payload, and local SMB/NetBIOS traffic, there were 2,588,729 "interesting" network connections. These form the input for the remaining analysis.

3.2 Results

Using 2.6 million network connections as input, the frequent string extraction step found 69,780 frequent strings. These frequent strings were then sorted using both the heuristic and the information theoretic ranking approach.

At this point, we needed to evaluate the ranking that our system produced. This turned out to be very challenging: When our system produces a token (snippet of characters), it is not trivial to determine whether this token is a good signature or not. Of course, there is also no ground truth available. After all, our system was developed with the goal of distinguishing C&C connections from benign traffic.

In a first step, we manually looked into the ranked list of candidate signatures. We focused on 100 entries on the top of the list, the middle, and the bottom, respectively. We then tried to understand whether a string was meaningful as a signature. To this end, we manually examined the payloads that were matched by each string. We then used our expert knowledge to make a judgment of whether the connections were likely malicious (hence, the signature is good), likely benign (the signature is not good), or suspicious (in this case, the signature likely matches bad traffic, but we could not confirm for certain). We found that many signatures in the top 100 correspond to malicious and suspicious traffic. Often, these signatures were quite specific and captured requests to certain web locations (confirmed as HTTP-based C&C), emails (spam), and IRC messages (confirmed as IRC-based C&C). In contrast, the bottom of the list mostly contained generic strings that are used by different applications. These include protocol headers (such as GET, POST, HTTP 1.1, etc.). An overview of results produced by this analysis process can be found in Table 1.

Table 1: Comparison with Autograph

	Our method			Autograph		
	good	benign	susp.	good	benign	susp.
Top 100	29	41	30	4	85	11
Middle 100	8	53	39	13	67	20
Bottom 100	8	56	37	11	41	38

3.3 Comparison with Prior Work

Autograph [13] is a well-known system for the automated extraction of signatures from network traffic. As mentioned previously, it relies on a pool of (mostly) malicious traffic and a pool of benign traffic. Unfortunately, in our case, all traffic is produced by malware, and we do not know *a priori* what are the C&C connections and what are unrelated, and ultimately benign, connections.

When using Autograph on our data set, the system basically sorts all strings based on their frequency. We manually checked the top 100, middle 100, and bottom 100 string produced by Autograph to determine which of these are likely good signatures, which are bad (i.e., they relate to benign traffic), and which are suspicious (maybe bad, but not obviously good either).

The result is shown on the right side in Table 1. The number of good signatures in the top 100 is 4, which is much lower than our results. This is because Autograph is designed to find the most frequent signatures first, whereas, in the case of C&C communication, being frequent is not a distinguishing property. Instead, scanning, spam, and exploit payloads are the most frequent signatures in the traffic.

We observed that only four out of the top 100 signatures, sorted based on their frequency, are related to C&C communications. This should be compared to the fact that 13 out of the top 20 signatures produced by our system are good (and at least 29 of the top 100). The reason for this behavior is that the C&C communication is occurring less frequently than scanning, exploitation, and spam payloads. Therefore, the most frequent strings correspond to such activities and not to C&C communication.

We also compared our top 100 signatures with Bothunter and a standard Snort installation (with the default signature set). In the comparison with Bothunter, we only used their ruleset for C&C detection (file `e4.rules` in the rules directory). Table 2 shows the results of running our system, Bothunter, and Snort on two data sets. The first data set was traffic gathered from a research lab in the computer science department of our university during one week of operation. This data set contains 400GB of network packets, produced by two dozen hosts. The second data set contained mostly malicious traffic, again recorded by the dynamic malware analysis system (but different from the set from which the signatures were extracted). Note that we found two signatures in Bothunter and two signatures among our signatures that produced a huge amount (more than a thousand) of false positives. We removed these four signatures from our analysis (this would be easy to do for a system administrator). As Table 2 shows, our approach had a better detection rate than both Snort and the Bothunter C&C detection signatures. Moreover, we did not produce any false positives on a week of this data set (after removing the two offending signatures).

3.4 Malware Family Classification

Previously, we made a claim that it is possible to use traffic clustering to group connections that belong to the same malware family. In particular, our system relies on the assumption that traffic

Table 2: Comparison with Bothunter and Snort

Data set	Our method		Bothunter		Snort	
	tp	fp	tp	fp	tp	fp.
Malicious traffic	6,766	0	2,223	0	85	11
Lab traffic	0	0	0	0	0	657

produced by a specific malware family is similar enough so that it will end up in the same cluster (or in very few clusters). In this section, we describe an experiment that supports this claim.

For this experiment, we first obtained the results from VirusTotal for each malware that produced network traffic. VirusTotal is an online service that accepts binary files, submits them to 42 anti-virus (AV) tools, and then returns the labels that each of these AV tools assigns to the binary. That is, VirusTotal outputs for a file the malware family labels generated by each AV tool. Unfortunately, each anti-virus program has its own convention for naming malware families, and often, labels for a single sample are contradictory. Thus, to obtain a single family name for each sample, we first removed the final (specific) part of each label, and then took the majority vote among all AV tools. This step produced (at most) one label for each sample.

Next, we discarded all families with less than 20 samples. This step resulted in 336 malware families. From each family, we randomly selected 20 samples. We then filtered out all samples that generated less than five network connections. Only 25 families remained for which each sample produced five or more connections.

For each sample that remained in our data set, we extracted the connection information and ran our clustering algorithm (described in Section 2.2.1). We then used the clustering results for classifying malware samples into 25 known malware families. More specifically, we used the RIPPER classifier [5] with 10-fold cross-validation. The classification achieved an accuracy of 56.8%. While this accuracy is clearly not sufficient for a detection tool, it supports our claim that connection clusters convey considerable amount of information about the malware families generating them. The experiment also demonstrated that network connections from the same malware family were grouped together in the same cluster. More specifically, more than 50% of the connections produced by each malware family were grouped in only one or two clusters.

It should be further noted that our achieved accuracy reflects a worst case scenario, as it has been negatively affected by several factors: First, the malware family names are not completely accurate as different AV tools use different naming standards for malware family names. Second, malware samples are run only for five minutes, which is a short period compared to the lifetime of a malware sample. Thus, a sample may not generate any network connections that are specific to its family in this period. Finally, we used only 20 samples for each malware class. A classifier accuracy can be improved by providing more training data. Despite these limitations, we found that traffic clusters reflect malware families quite accurately. This is important so that our system can extract meaningful command and control strings.

3.5 Limitations

There are three main limitations of the proposed approach. First, the system can be used only when the C&C traffic is not encrypted. If the C&C traffic is encrypted, content-based signatures cannot be extracted. This problem is shared by all systems that use signatures that operate on the traffic content. Second, the system cannot provide a guarantee that all strings that receive high scores are indeed related to C&C traffic. Thus, a certain amount of involvement from a human analyst is beneficial to remove irrelevant strings. However, our system reduces the manual effort significantly by focusing the analysis effort on *likely* C&C candidate signatures. Also, one could further remove possibly irrelevant strings by matching the results against benign traffic, discarding all strings that trigger too frequently on benign connections.

4. RELATED WORK

There has been a considerable amount of work on how to detect and mitigate botnets. Bailey et al. classified botnet detection techniques into three classes [1]: *1- Detecting cooperative behavior:* One intrinsic aspect of botnet activity is that the machines in a botnet cooperate towards the same goal. This cooperation could be detected by various correlation techniques. *2- Detecting by signature:* These approaches are based on detecting botnets using some predefined patterns of activity. *3- Detecting by attack behavior:* Detecting botnets by detecting the patterns of the attacks botnets perform.

Correlation-based detection. A number of previous systems focus on the large-scale, correlated behavior of bots. For example, Gu et al. presented BotSniffer [11], which uses the temporal correlation between the activity of infected machines to detect botnets. In follow-up work, Gu et al. used correlation between connection behavior (C-plane) and attack behavior (A-plane) [8]. Strayer et al. used correlation between traffic characteristics of different IRC connections to identify botnets [23]. Zhao et al. presented a correlation-based approach to detect webmail user accounts used by botnets [27]. Coskun et al. proposed graph-based algorithm to detect P2P bots [6]. The problem with correlation approaches is they require access to a large-scale network with many active bots. Moreover, there is often a non-trivial amount of time that passes before enough evidence of a bot infection is collected.

Signature-based detection. Some previous work attempts to distinguish botnet IRC command and control messages from legitimate, real-user IRC communications. Livadas et al. used naïve Bayesian networks to distinguish between real IRC communication and the one used by bots [17]. Gu et al. used an active probing approach to intervene in ongoing suspicious IRC communications to distinguish a human conversation from a bot conversation [10]. Kondo et al. used SVM techniques on packet characteristics to differentiate between legitimate IRC traffic and IRC C&C channels [14]. The problem with these approaches is that they try to find C&C communication only in IRC communication, while botnets are actively using other types of protocols to carry their C&C messages. Therefore, a more general C&C detection method is needed. Bilge et al. [2] use machine learning to train classifiers to detect C&C servers.

Gu et al. proposed a more-general bot lifecycle model, and they tried to detect botnets by correlating the events that represent different stages of this lifecycle [9]. At its core, this system, together with more traditional, misuse-based intrusion detection systems, relies on a database of signatures that identify C&C botnet traffic. Unfortunately, these approaches suffer from the same problems that all misuse-based detection mechanisms have: When malicious software changes its behavior, it will no longer be detected. In most cases, signatures are generated manually, and this is the process that our work seeks to automate.

The work closest to ours is the system presented by Wurzinger et al. [25]. The authors introduce a system for automatically generating botnet C&C signatures, and leverage the fact that, after receiving a command, the behavior of the bot changes considerably. This change can be detected using network behavior analysis. Wurzinger et al. then look into the traffic seen before this change of behavior to extract possible commands. While effective, this approach can only detect commands that lead to significant changes

in the bots' network behavior, such as commands that trigger denial of service attacks or spam runs. Commands that results in less obvious reactions, such as data leakage or "phoning home," cannot be extracted.

Automated signature generation. There has been a significant amount of previous work on automated signature generation for Internet worms [3, 4, 13, 15, 19, 21, 28]. As previously mentioned, the key difference is that these systems are provided with a pool of traffic that contains mostly malicious connections. The goal is simply to find strings that cover this pool without matching too many legitimate connections. Unfortunately, we do not know *a priori* which of the connections in our input are C&C traffic. Hence, we need to introduce a ranking function that assigns a higher score to likely C&C signatures.

Perdisci et al. [20] propose a system to extract signatures for HTTP-based bots. To this end, the authors first cluster similar HTTP requests. Then, for each cluster, a generalized signature is extracted. The difference to our approach is twofold. First, the system by Perdisci et al. only works for HTTP-based C&C traffic. Second, the system cannot distinguish between signatures that match HTTP-based command and control traffic and HTTP connections that are unrelated to C&C connections. Thus, the system produces signatures for all non-C&C traffic as well. Our system, on the other hand, introduces a ranking function that will assign a higher score to signatures capturing C&C traffic.

5. CONCLUSIONS

In this paper, we presented a novel approach for extracting C&C signatures. Our approach automates the tedious, manual work currently necessary when deriving botnet signatures for network intrusion detection systems. Our system takes as input a pool of network connections that are produced by running bot samples in Anubis. The system first extracts all frequent strings from the traffic. Then, a ranking function is invoked that assigns a high score to traffic-class distinguishing strings. These strings are likely good C&C signatures. The reason is that C&C connections for a botnet, or an individual command of a botnet, share network-level similarities, while non-C&C traffic is more diverse. We applied our approach to traffic collected by a dynamic malware analysis system. Furthermore, we showed that our method has a better accuracy than Autograph. We also compared detection accuracy of our C&C signatures with those of Bothunter and Snort. The system analyzed almost 2.6 million connections produced by more than 1.4 million samples. The results demonstrate that our system has real-world applicability and can extract meaningful C&C signatures.

6. ACKNOWLEDGMENTS

This work was supported by the Office of Naval Research (ONR) under Grant N000140911042, the Army Research Office (ARO) under grant W911NF0910553, the National Science Foundation (NSF) under grants CNS-0845559 and CNS-0905537, and Secure Business Austria.

7. REFERENCES

[1] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir. A Survey of Botnet Technology and Defenses. In *CATCH '09: Proceedings of the 2009 Cybersecurity Applications & Technology Conference for Homeland Security*, pages 299–304, Washington, DC, USA, 2009. IEEE Computer Society.

[2] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel. Disclosure: detecting botnet command and control servers through large-scale netflow analysis. In *Proceedings of the 28th Annual Computer Security Applications Conference*, pages 129–138. ACM, 2012.

[3] M. Cai, K. Hwang, J. Pan, and C. Papadopoulos. WormShield: Fast Worm Signature Generation with Distributed Fingerprint Aggregation. *IEEE Trans. Dependable Secur. Comput.*, 4(2):88–104, 2007.

[4] L. Cavallaro, A. Lanzi, L. Mayer, and M. Monga. LISABETH: automated content-based signature generator for zero-day polymorphic worms. In *SESS '08: Proceedings of the fourth international workshop on Software engineering for secure systems*, pages 41–48, New York, NY, USA, 2008. ACM.

[5] W. W. Cohen. Fast effective rule induction. In *In Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann, 1995.

[6] B. Coskun, S. Dietrich, and N. Memon. Friends of an enemy: identifying local members of peer-to-peer botnets using mutual contacts. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, pages 131–140, New York, NY, USA, 2010. ACM.

[7] R. M. Gray. *Entropy and information theory*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[8] G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *SS'08: Proceedings of the 17th Conference on Security symposium*, pages 139–154, Berkeley, CA, USA, 2008. USENIX Association.

[9] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: detecting malware infection through IDS-driven dialog correlation. In *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 1–16, Berkeley, CA, USA, 2007.

[10] G. Gu, V. Yegneswaran, P. Porras, J. Stoll, and W. Lee. Active Botnet Probing to Identify Obscure Command and Control Channels. In *Proceedings of 2009 Annual Computer Security Applications Conference (ACSAC'09)*, December 2009.

[11] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *In Proceedings of 15th Annual Network and Distributed System Security Symposium (NDSS 2008)*, 2008.

[12] X. Ji, J. Bailey, and G. Dong. Mining minimal distinguishing subsequence patterns with gap constraints. *Knowl. Inf. Syst.*, 11(3):259–286, 2007.

[13] H.-A. Kim and B. Karp. Autograph: toward automated, distributed worm signature detection. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, pages 19–19, Berkeley, CA, USA, 2004. USENIX Association.

[14] S. Kondo and N. Sato. Botnet Traffic Detection Techniques by C&C Session Classification Using SVM. In *In Proceedings of IWSEC*, pages 91–104. 2007.

[15] C. Kreibich and J. Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *SIGCOMM Comput. Commun. Rev.*, 34(1), 2004.

[16] C. Kruegel, E. Kirda, and U. Bayer. Ttanalyze: A tool for analyzing malware. In *Proceedings of the 15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*, 4 2006. Best Paper Award.

[17] C. Livadas, R. Walsh, D. Lapsley, and W. T. Strayer. Using Machine Learning Techniques to Identify Botnet Traffic. In *In 2nd IEEE LCN Workshop on Network Security (WoNS '2006)*, pages 967–974, 2006.

[18] T. M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1997.

[19] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *SP '05: Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 226–241, Washington, DC, USA, 2005. IEEE Computer Society.

[20] R. Perdisci and W. Lee. Behavioral clustering of http-based malware and signature generation using malicious network traces. *of the 7th USENIX conference on*, 2010.

[21] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 4–4, Berkeley, CA, USA, 2004. USENIX Association.

[22] 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 *CCS '09: Proceedings of the 16th ACM conference on Computer and communications security*, New York, NY, USA, 2009. ACM.

[23] 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 (LCN)*, pages 15–16, 2006.

[24] E. Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14:249–260, 1995. 10.1007/BF01206331.

[25] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically generating models for botnet detection. In *In 14th European Symposium on Research in Computer Security (ESORICS)*, 2009.

[26] O. Zamir and O. Etzioni. Web document clustering: a feasibility demonstration. In *SIGIR '98: Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 46–54, New York, NY, USA, 1998. ACM.

[27] Y. Zhao, Y. Xie, F. Yu, Q. Ke, Y. Yu, Y. Chen, and E. Gillum. BotGraph: large scale spamming botnet detection. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 321–334, Berkeley, CA, USA, 2009. USENIX Association.

[28] U. Zurutuza, R. Uribeetxeberria, and D. Zamboni. A data mining approach for analysis of worm activity through automatic signature generation. In *AISec '08: Proceedings of the 1st ACM workshop on Workshop on AISec*, pages 61–70, New York, NY, USA, 2008. ACM.