On Botnets that use DNS for Command and Control

Christian J. Dietrich[†‡], Christian Rossow[*‡], Felix C. Freiling[†],
Herbert Bos[*], Maarten van Steen[*] and Norbert Pohlmann[‡]

[*]*Computer Systems Group*
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands

[†]*Department of Computer Science*
Friedrich-Alexander University
Erlangen, Germany

[‡]*Institute for Internet Security*
University of Applied Sciences Gelsenkirchen
Gelsenkirchen, Germany

Abstract

We discovered and reverse engineered Feederbot, a botnet that uses DNS as carrier for its command and control. Using k-Means clustering and a Euclidean Distance based classifier, we correctly classified more than 14m DNS transactions of 42,143 malware samples concerning DNS-C&C usage, revealing another bot family with DNS C&C. In addition, we correctly detected DNS C&C in mixed office workstation network traffic.

***Index Terms*—malware detection; botnet detection; dns; command and control;**

I. Introduction

Botnets, i.e. sets of computers that are infected with a specific malicious software that allows these computers to be remote controlled, have become one of the biggest security issues on the Internet imposing a variety of threats to Internet users. Therefore, organizations have keen interest to keep the number of bot infections low. Since the remote command and control channel (C&C) is a defining characteristic of botnets, techniques have been developed to *detect* bot infections by identifying the C&C network traffic. This has been (partly) successful, e.g. for IRC- [1] or HTTP-based botnets [2].

Advances in malware research have challenged botnet operators to improve the resilience of their C&C traffic. Partly, this has been achieved by moving towards decentralized structures (like P2P) or by otherwise obfuscating and even encrypting communication [3–8]. This makes it harder for researchers to distinguish malicious from benign traffic, albeit not impossible. It was only a question of time when botnet C&C channels would be designed in a way that C&C messages are hidden in common application layer protocols, striving for covert communication.

Recently, we observed a specific type of malware termed *Feederbot* that showed strange behavior in the sense that it seemingly did not use any obvious C&C channel. By reverse engineering the particular sample, we found out that the bot (ab)used DNS as a communication channel for C&C traffic. Apart from this insight, we were interested in the difficulties to detect this type of seemingly "covert" and "hard to detect" traffic. Since DNS has not been documented so far as a C&C protocol in botnets[1], such botnets benefit from the fact that currently there is no specifically tailored detection mechanism, which in turn raises the probability for the botnet to remain undetected. We achieved to detect this particular type of C&C traffic using machine learning techniques and traffic analysis.

For example, we applied the resulting method on purely malicious traffic produced using our dynamic malware analysis network Sandnet [10] and found that in over 14 million DNS transactions of over 42,000 malware binaries we did not produce any false positive. In fact, in addition to Feederbot, we were able to identify a second class of malware that also used DNS as C&C channel.

One reason for our good results was the way in which DNS was used for communication: the botnet was using the technique of *DNS tunneling* to evade detection. DNS tunneling refers to the technique in which data is transmitted within (resource record) fields of a DNS message. As a bottom line, our results underline that covert communication must not necessarily be harder to detect than non-covert communication. On the contrary, the covert communication we analyzed introduced anomalies to DNS traffic that can be identified. So the difficulty was not only to detect the presence of C&C information in DNS, it was also to identify the carrier (i.e. DNS) over which covert communication takes place.

In summary, the contributions of this paper are threefold:

- To our knowledge, we are the first to document

[1]There is only anecdotal evidence for DNS as botnet C&C [9].

DNS-based botnet C&C traffic.

- We present a technique that distinguishes between DNS-based C&C and regular DNS communication in real-world DNS traffic. In other words, we provide a technique for the *detection* of this particular class of malware.
- We present a classifier that can distinguish purely malicious communication into DNS-based C&C and regular DNS communication. In other words, we provide a technique for the *classification* of malware samples based on their behavior.

This paper is structured as follows: We present the case study of Feederbot in Section II. We then describe our detection and classification approach in Section III. We give a brief discussion of our findings in Section IV and describe related work in Section V.

II. Case Study: DNS as Botnet C&C

From the point of view of a botmaster, a trade-off between C&C communication visibility and the bot-inherent need to communicate arises. On the one hand, bots must communicate with their C&C instance to receive instructions and transmit data such as stolen credentials. On the other hand, botmasters try to hide the C&C traffic in order to avoid detection. Usually, the design of a botnet's C&C results in messages being obfuscated or encrypted so that it is more difficult to detect and understand the semantics of certain types of C&C traffic.

A. *DNS as carrier for Botnet C&C*

Whereas several application layer protocols have been analyzed by the research community concerning the usage as a basis for botnet command and control, to our knowledge we are the first to openly analyze the Domain Name System protocol as carrier for botnet C&C. DNS, when compared to other application layer protocols provides some advantages. Concerning its usage as botnet C&C, DNS has not been seen so far. Thus, botnets using DNS as C&C benefit from the fact that currently there is no specifically tailored detection mechanism, which in turn, raises the probability for the botnet to remain undetected. Even in environments with heavily restricted Internet access, e.g. by means of firewalling and proxying, DNS is usually one of the few protocols – if not the only one – that is allowed to pass without further ado. Furthermore, whereas for some protocols such as HTTP, there are a number of existing methods to analyze and inspect the network traffic like the one presented by Perdisci et al. [2], DNS is usually served "as is". As another advantage, DNS was designed as a distributed system and as such provides advantages in terms of resilience.

Using our dynamic malware analysis environment called Sandnet [10], we discovered a bot that indeed uses DNS messages as carrier for command and control traffic. As DNS is a new kind of botnet C&C, we provide some insight into the inner workings of this bot named *Feederbot*. We gained insight by reverse engineering the Feederbot sample as well as analyzing the network traffic that was captured during the analysis of Feederbot in Sandnet.

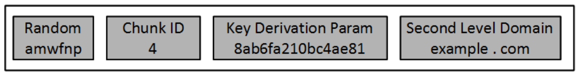

Fig. 1. *Example of Feederbot's DNS Query Domain Name*

Feederbot uses valid DNS syntax. Its C&C messages consist of DNS messages with TXT resource records. Furthermore, the query domain name is used to transmit certain parameters from the bot to the C&C server such as parameters for key derivation. An example of the query domain name structure is given in Figure 1. Feederbot has to query the C&C servers directly, bypassing the pre-configured DNS resolver on the host because the domains that are used in Feederbot's requests are not delegated. Manual resolution of seven domain names seen in Feederbot requests starting at the DNS root, i.e. not querying Feederbot's DNS C&C servers, results in NXDOMAIN responses.

We can only speculate as to why Feederbot avoids the pre-configured resolver and directly queries its DNS servers. One reason could be that in this way, the corresponding DNS C&C transactions leave no traces in DNS resolver logs, caches or passive DNS databases. In contrast, the fact that a different than the pre-configured DNS resolver is used, might itself be suspicious enough to catch one's eye – especially in homogeneous environments.

B. *Segmentation and Encryption*

Feederbot's C&C traffic is split into message chunks with a maximum length of 220 bytes per chunk. One message chunk is transmitted in the rdata field of a TXT resource record in the DNS response. The structure of a Feederbot message chunk is shown in Figure 2. The query domain name (Figure 1) contains among others the identifier for the message chunk that is to be retrieved from the C&C server.

In order to evade detection, most of the message chunks are encrypted using the stream cipher RC4. Feederbot uses a variety of different encryption keys. A specific part of the DNS query domain name is used to transmit parameters for key derivation. As an example, one such parametrized key derivation function takes as input a substring of the query domain name $qdparam$.

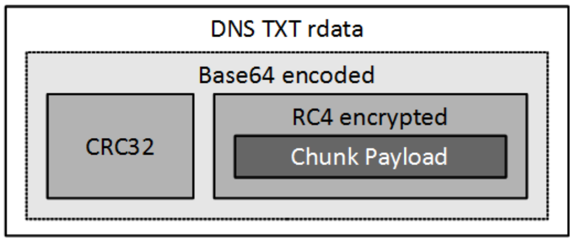

Fig. 2. *Structure of a Feederbot DNS C&C Message Chunk*

This substring $qdparam$ is then RC4-encrypted with the string “feedme” and the result is used to initialize the RC4 decryption of the actual C&C message chunks. The stream cipher is used in a stateful manner, so that if a message chunk gets lost, decryption of subsequent message chunks will fail. In addition, Feederbot’s C&C message chunks make use of cyclic redundancy checks to verify the decryption result. The CRC32 checksum preceeds message chunk payload and is not encrypted.

Using the results from the dynamic analysis in Sandnet as well as the reverse engineering efforts, we achieved the implementation of a passive decryption utility in order to decrypt Feederbot’s DNS C&C messages. Additionally, we implemented a low interaction clone of Feederbot to actively analyze its C&C.

Feederbot receives instructions from several DNS C&C servers. Initially, when Feederbot is launched, a request is sent to a very small subset of C&C servers. These servers seem to serve as bootstrapping C&C servers. During our monitoring period of nine months, we have seen only two such DNS servers (in terms of IP addresses) acting as bootstrapping C&C servers. The response to this initial bootstrapping request message is not encrypted and only base64 encoded. It contains a pointer to at least one other C&C server as well as another domain name. Subsequent communication is encrypted.

III. Detecting DNS-based Botnet C&C

When facing a new kind of botnet C&C as with Feederbot, it is of concern how such a communication can be detected. Inspired by the results of the Feederbot analysis, we developed machine learning features for a DNS C&C classification method.

Our classification approach is based on several prerequisites. First, we assume that DNS-based C&C channels typically carry dense information, such as compressed or encrypted data. As botmasters strive for resilience, encryption is becoming more prevalent among botnet C&C. In addition, due to the message length limits of DNS, botmasters need to fit their commands into relatively small messages. Second, we assume that to a certain degree there is a continuous “downstream” flow of information, i.e. from the C&C server to the bot. Admittedly, this second condition may not be met for bots with certain damage functionalities, e.g. low profile trojans. However, we believe that lots of bots actually fulfil this requirement, especially spam-sending bots or click fraudsters. In these cases, the C&C server is required to continuously provide the bot with input data, such as spam target email addresses, templates and text blocks or URLs to feed the click fraud module.

A. Classification Features

Key to our detection method are certain differences between regular DNS usage and DNS C&C, which we divide in two categories. The first category deals with differences concerning the use of the rdata field whereas the second category addresses differences concerning the communication behavior.

1) Rdata Features: The basic unit for these entropy-based features is the rdata field of all resource records of one DNS response. For brevity, this unit is referred to as rdata message in the following. As an example, in the context of Feederbot this corresponds to one message chunk. Note that we do not restrict the features to certain resource record types or sections.

Shannon entropy is a measure of randomness in a string of data. Given a finite alphabet $\Sigma = \{0, 1..255\}$, the entropy estimates how randomly the characters in word w are distributed. We use the maximum likelihood estimator to calculate the *sample entropy* of a message $w \in \Sigma^*$, f_i denotes the frequency of character i:

$$\hat{H}(w) \;=\; -\sum_{i=0}^{255} f_i \cdot log_2(f_i)$$

Then, the word $w_1 = 001101$ has a lower sample entropy than the word $w_2 = 012345$. We exploit the fact that encrypted or compressed messages have a high entropy. As we assume encrypted C&C, the C&C messages exhibit a high entropy. Encrypted data composed of characters of the full 8-bit-per-byte alphabet will converge towards the theoretical maximum entropy of 8 bits per byte. In this case, entropy is typically referred to as *byte entropy*. In fact, when using DNS as C&C, certain fields of the DNS protocol such as TXT or CNAME resource records’ rdata do not allow the full 8 bits to be used per byte. Thus, botmasters have to “downsample” their C&C messages to the destined alphabet, e.g. by means of Base64 or Base32 encoding. This implies that the resulting message exhibits a comparatively low *byte entropy*. We overcome this issue by estimating the destined alphabet size by counting the number of distinct characters in a given field. After that, we calculate the expected sample entropy for random data based on the estimated alphabet size.

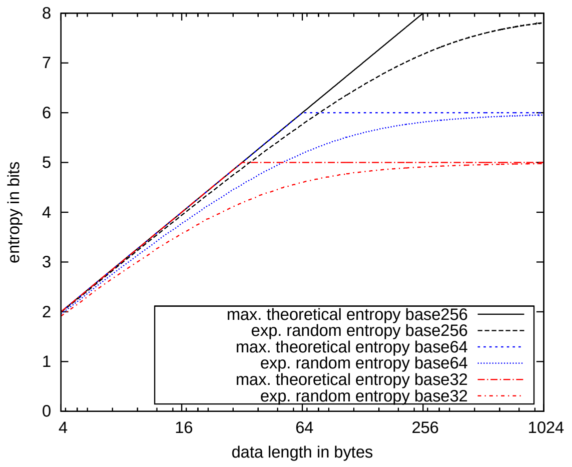

Fig. 3. Statistical byte entropy

Another issue is posed by the fact that short strings of data – even when composed of random characters – rarely reach the theoretical maximum entropy. For example, a string of 64 bytes length, based on the 8-bit per byte alphabet Σ has a theoretical maximum byte entropy of 6 bits. However, considering a string r of 64 bytes length with randomly distributed bytes of the alphabet Σ, the byte entropy is typically lower than 6 bits, e.g. around 5.8 bits. This finding is based on the birthday paradox. Basically, encrypted data is randomly distributed, but randomness does not imply a uniform distribution. Thus, if a string r is short (e.g. 64 bytes), the expected byte entropy is significantly below 8 bits, although r might be purely random.

We overcome this issue by calculating the *statistical byte entropy* for a string of a given length. This is done as follows. Empirically, we compute the average byte entropy of a set of $x = 1,000$ random words for every length $1 < N < 2^{10}$. For any word $w_1, \ldots, w_x$, we compose a random byte distribution and calculate the byte entropy. Since x was chosen sufficiently large, calculating the mean over all x byte entropies of words with length N estimates the expected *statistical byte entropy* of random data of length N. Figure 3 shows the maximum theoretical entropy and the expected statistical random entropy for the full 8-bit per byte alphabet Σ as well as typical Base64 and Base32 alphabets. One important feature is the deviation of the actual sample entropy to the expected statistical random entropy. Effectively, this covers different alphabets in a flexible fashion, i.e. we can dynamically detect high sample entropies in data encoded with any alphabet such as Base64, Base32, Base16 or the like.

In addition, we measure the minimum and maximum byte values of a given rdata message as well as the coverage of some subsets of the ASCII character set such as capital letters and digits. The complete list of features for an rdata message m consists of:

- number of distinct byte values in m
- minimum byte value in m
- maximum byte value in m
- number of ASCII capital letters (byte values 65-90) in m
- number of ASCII digits (byte values 48-57) in m
- length of m in bytes
- absolute difference of the statistical byte entropy at given length of m and the entropy of m

2) Aggregated Behavioral Communication Features: Furthermore, we exploit behavioral properties of DNS C&C bots. These features address the fact that a certain amount of information has to be transmitted via the DNS C&C channel, e.g. spam templates, email addresses or click fraud URLs, and that the C&C channel exhibits a certain level of persistence. In contrast to the rdata features, the behavioral communication features do not operate on individual DNS response resource records. Instead, they operate on aggregated data. Thus, for the behavioral communication features to be computed, we need to define aggregation criteria, so that communication properties can be aggregated over time before calculating the features. When analyzing Feederbot, we observed that the DNS C&C servers were contacted directly, avoiding the pre-configured DNS resolver. In this case, in order to compute behavioral communication features (e.g. bandwidth), the requester's IP address as well as the IP address of the DNS server serve as aggregation criteria. In contrast, if DNS requests are directed towards pre-configured DNS resolvers, the requester's IP address and the query domain name (and parts thereof such as the second-level domain) are used as aggregation criteria. In both cases, we assume that the requester's IP address represents one host, i.e. no network address translation has been performed. Note that even if network address translation was performed, detecting that DNS C&C was used might still be possible. However, in such a case, it would be impossible to identify the infected host.

We define the following behavioral communication features. First, we measure the size of all rdata messages (i.e. rdata fields of DNS response resource records) and compute the corresponding aggregated bandwidth over time. We expect that the data volume transmitted between the DNS C&C server and the bot will tend to be significantly larger when compared to regular DNS usage, as DNS is typically not used for data transmission. This observation results in either larger messages and/or in an increased bandwidth consumption between the bot and the C&C server.

Second, the information flow between one bot instance and the C&C server is expected to appear more persistent. We measure the persistence as the maximum of the time between two DNS responses, as well as the communication duration calculated as the time between the first and the last message exchanged with a C&C server. Yet simple, we expect these behavioral communication features to be effective enough in order to extend a classifier based on the rdata features.

B. Clustering DNS traffic

Given these features, our goal is to develop a binary classifier that is able to detect DNS-C&C traffic. Before we can classify DNS traffic, we need to extract two sets of training data. As a first step, we extract two different kinds of DNS traffic. We define one transaction to be composed of one DNS request as well as the corresponding DNS response. Based on the network traffic caused by the Feederbot execution that we analyzed in Section II, we compile a set of known Feederbot DNS C&C transactions. This set is referred to as D_D and contains 3128 DNS transactions. D_D is composed of DNS transactions fulfilling both of the following two conditions:

- The transaction was directed to any of the DNS bootstrapping C&C servers which were verified to be used as C&C during reverse engineering.
- The request has a query domain name ending in one of 7 second-level domains observed during dynamic execution analysis.

Furthermore, we extract D_N, a set of DNS transactions of 30 executions of bots that knowingly do *not* use DNS as C&C. In addition, we manually inspected 500 (1%) of the 47,433 DNS transaction of D_N. D_N contains DNS transactions that occurred as part of the damage functionality of these bots such as spamming or click fraud. The complete list of bot families used to compile D_N is given in Table I. The bot family names are based on a majority voting of up to 43 labels per sample.

Bot Family	*Type of C&C*	*# Execs*	*DNS TXs*
unknown	HTTP	3	620
unknown	IRC	4	1951
agobot	IRC	1	163
koobface	HTTP	2	4119
rbot	IRC	2	300
sality	Custom P2P	4	5718
sdbot	IRC	3	916
swizzor	IRC	1	93
virut	IRC+CE	4	17,740
virut	IRC (plaintext)	4	15,789
zbot	HTTP+CE	2	24

TABLE I
BOT EXECUTIONS USED TO ACQUIRE NON-DNS-C&C TRANSACTIONS. CE=CUSTOM ENCRYPTION

With respect to approximately equally sized training sets, the next step consists in drawing 5000 elements of D_N at random into D_{NS} so that the resulting set D_{NS} has approximately the same size as D_D. We compose the set $D := D_D \cup D_{NS}$ as the union of D_D and D_{NS}.

At this point, we extract the rdata features described in Section III-A1 from all DNS transactions in D. The resulting set of feature vectors is referred to as F. Moreover, the elements of F are normalized and the normalization parameters are stored. Using k-Means clustering with $k = 2$ and Euclidean Distance function, we separated F into two clusters C_D and C_N. The cluster which contains the most known DNS C&C elements is considered as C_D, the other one as C_N.

The clustering step aims at distilling the characteristic transactions for DNS C&C into the resulting cluster C_D. Table II, the classes to clusters comparison shows that only 6 elements of D_D were assigned to cluster C_N. Manual inspection revealed that each of these 6 transactions carries a Feederbot C&C message chunk with an empty payload. Thus, these are not considered as characteristic C&C messages. All of the transactions in the Non-DNS-C&C set D_{NS} were assigned to the Non-DNS-C&C cluster C_N.

	C_N	C_D
D_{NS}	5000	0
D_D	6	3122

TABLE II
CLASSES TO CLUSTERS COMPARISON

Based on the clustering results, we develop a classification method. First, we aim at classifying malicious network traffic, i.e. network traffic caused by malware as it is acquired in Sandnet. This binary DNS traffic classifier is supposed to distinguish between DNS-based C&C and Non-C&C in order to find other malware executions that exhibit DNS C&C. Second, we aim at detecting DNS-based C&C channels in real-world DNS traffic.

C. Detecting Bots that use DNS C&C

As we were curious to find further malware samples using DNS C&C – apart from the Feederbot sample and its execution that we analyzed in depth in Section II – we designed a DNS C&C classifier that can be applied to the network traffic gained by our Sandnet analysis of more than 100,000 malware samples [10]. Due to the enormous amount of data and because we wanted to identify individual DNS C&C transactions, we intentionally restrict ourselves to the rdata features as described in Section III-A1. Therefore, we calculate the rdata features for the 14,541,721 DNS transactions of all 42,143 samples that have been executed in Sandnet and that exhibited DNS traffic. Each feature vector reflects one DNS transaction.

In order to classify DNS traffic, we calculate the mean cluster centroids of both clusters C_D and C_N built in Section III-B. Each feature vector is scaled using the normalization parameters from the training phase. Finally, as classification method, we implemented a Euclidean Distance based classifier which assigns the class of the closest cluster to the given feature vector.

All in all, 109,412 DNS transactions of Sandnet traffic are classified as DNS C&C. This procedure reveals 103 further executions of Feederbot samples. Surprisingly, our classification even discovers another bot family that uses DNS-based C&C. We term this newly found bot *Timestamper* due to the fact that it uses the Unix timestamp of the current date and time in the query domain name. Timestamper, in contrast to Feederbot, uses the preconfigured DNS resolver. Our classifier detects 53 executions of Timestamper in Sandnet data. Manual inspection verifies that all of the 156 executions which have associated DNS transactions classified as DNS C&C are either Feederbot or Timestamper executions. Thus, at this point, we draw the conclusion that, in terms of Sandnet executions, our classifier does not produce any false positive.

As part of an effort to estimate the False Negative rate, we compile a regular expression for Timestamper's DNS C&C requests that matches the Unix timestamp in the query domain name. 1679 transactions where the regular expression matches the query domain name are considered as Timestamper's DNS C&C requests, the remaining 1851 DNS transactions are considered as Non-C&C DNS. Our classifier correctly classifies all of the 1679 transactions as being DNS C&C transactions, i.e. showing no false negatives among Timestamper's DNS C&C traffic.

In addition, we evaluate our classifier against 1851 Timestamper DNS transactions which are *not* part of its DNS C&C in order to estimate the False Positive rate on the transaction level. Once more, our classifier correctly considers all of these 1851 transactions as not being DNS C&C transactions, i.e. showing no false positive among the Timestamper DNS transactions.

To summarize, our binary DNS C&C transaction classifier successfully reveals 103 further executions of Feederbot and discloses 53 executions of Timestamper, the newly disclosed bot that also uses DNS C&C. In addition, the results show that even though we trained only on known Feederbot DNS C&C of one execution, our classifier was able to correctly classify DNS C&C transactions of another, completely unrelated type of malware.

D. Detecting DNS C&C in mixed traffic

Furthermore, we evaluate our classifier on mixed workstation DNS traffic. Therefore, we recorded DNS network traffic at our Institute at the network router/NATting point where all traffic from workstations towards internal servers as well as arbitrary Internet destinations passes. Traffic from the internal servers heading for Internet destinations was excluded in order for recursive DNS queries caused by the DNS resolver to be avoided. All DNS traffic was recorded before source network address translation (NAT) on the router was applied. In this manner, we are able to capture DNS traffic destined for the pre-configured DNS resolver and for remote DNS resolvers on the Internet.

Additionally, we executed one Feederbot sample and one Timestamper sample from inside the workstation net, each for one hour. Both samples were executed in virtual machines on workstation computers that were used for regular operation throughout the whole measuring period. The network access of the virtual machines was configured to use NAT on the workstation hosts. Thus, the traffic originating in each infected virtual machine and the corresponding workstation host traffic cannot be distinguished by source IP address and regarding our aggregation they represent one entity. Additionally, all network traffic caused by the virtual machines was recorded individually on the workstation host. Our goal is to detect those workstations that executed the Feederbot and Timestamper samples.

The captured network traffic contains a total of 69,820 successful DNS transactions from 49 distinct workstation IP addresses of our Institute, captured between 7 a.m. and 8 p.m. on a regular weekday. This dataset is referred to as T_{all}. The Feederbot VM caused a total of 2814 DNS transactions (T_F) among which 1092 were DNS C&C transactions (T_{FCnC}). Additionally, we observed 4334 HTTP flows during its click fraud activity. The network trace of the VM executing the Timestamp bot showed a total of 181 DNS transactions (T_T) with 102 DNS C&C transactions (T_{TCnC}). Consequently, the traffic capture contains 66,825 DNS transactions caused by the legitimate workstations during regular operation.

In order to make use of the aggregated behavioral communication features, we extended our classification method. Based on the results in Section III-C, we compiled a set of 10 Feederbot executions revealed in Section III-C. For these executions, we calculated the following three thresholds:

1) t_b the mean bandwidth per aggregate
2) t_{mi} the mean of the maxima of the gaps between two consecutive C&C messages for DNS C&C flows

3) t_{si} the standard deviation of the maxima of the gaps between two consecutive C&C messages for DNS C&C flows

As a first step, we applied the classifier presented in Section III-C to all of the DNS transactions in T_{all}. This results in the set of candidate DNS C&C transactions T_{cand}. Furthermore, in order to apply the behavioral communication features, we computed two kinds of aggregates for the candidate transactions in T_{cand}. First, we aggregate by each pair of source and destination IP addresses. Second, we aggregate by each pair of source IP address and second level domain of the query domain name.

Subsequently, the set of aggregates is filtered, eliminating all aggregates that do not fulfil the behavioral properties. We exclude aggregates with a computed bandwidth smaller than t_b and a maximum time between two C&C messages greater than $d \cdot t_{si} + t_{mi}$ with $d = 3$. This filtering step makes sure that only those channels with persistence be considered as C&C channels. None of the aggregates were excluded in the filtering step.

Based on the resulting set of aggregates, we consider each source IP address to be infected with malware using DNS C&C. Indeed, only the two IP addresses of the workstations that hosted the Feederbot and Timestamper bots were classified as DNS C&C infected hosts. To sum up, we showed that our classifier can even detect DNS C&C transactions in mixed network traffic of regular workstations.

IV. Discussion

Though achieving high true positive rates, there are certain limitations that bots could exploit to evade our detection. One such limitation is posed by the fact that botmasters could restrict their C&C messages to very small sizes. In practice, message contents could be stored in e.g. 4 bytes of an A resource record's rdata. In this case, our rdata features alone, which are currently applied to individual C&C messages, would not be able to detect these C&C messages as high entropy messages because the statistical byte entropy of really short messages is very low and our estimate of the alphabet size by counting the number of distinct bytes is inaccurate for short messages.

In this case, a countermeasure could be to aggregate several messages and compute aggregated rdata features. Furthermore, for each aggregate, the change of entropy among subsequent messages can be measured. Additionally, for certain resource records one could compare the distribution of byte values against the expected distribution. For example, the rdata of an A resource record contains IPv4 addresses. However, the IPv4 address space is not uniformly distributed. Instead, certain IPv4 address ranges remain reserved, e.g. for private use such as 10.0.0.0/8 (RFC1918) or 224.0.0.0/4 for multicast. These might rarely show up in Internet DNS traffic whereas other addresses, e.g. popular web sites, might appear more often in DNS query results.

When looking at Feederbot, it becomes obvious that the query domain name can be chosen completely at random. In general, this is true for botnets where the DNS C&C servers are contacted directly. In order to avoid raising suspicion, the botmasters could have chosen e.g. random or even popular second-level domains. This would become a problem for our detection mechanism if only the query domain name was used for aggregation alone. However, as we also aggregate by the DNS server's IP address, our classifier can still detect this kind of DNS C&C. As a result, we suggest to aggregate by at least both, the DNS server's IP address and the query domain name, because the botmaster can only arbitrarily change one of them.

Another limitation is posed by the fact that our behavioral communication features aim at botnets with a central C&C architecture using a limited set of C&C servers. Botmasters might exploit this by spreading the communication with C&C servers over lots of different C&C destinations so that even the aggregated behavioral features such as the aggregated bandwidth remain subliminal. Effectively, a further step in this direction would be to change for a peer-to-peer C&C architecture where each bot is part of the peer-to-peer network. However, this opens the door for a whole variety of techniques addressing peer-to-peer networks such as eclipse attacks.

V. Related Work

Related work can be grouped in three categories. First, several C&C techniques of botnets have been analyzed in depth [3–5, 7, 11]. For example, Holz et al. [5] provide insights into botnets with peer-to-peer C&C architecture in a case study on the storm worm. Stock et al. and Calvet et al. [3, 11] analyze the Waledac peer-to-peer botnet in detail. However, to our knowledge, we are the first to analyze DNS as carrier for botnet C&C. Our work depicts how C&C messages are structured, encrypted and encoded in regular DNS syntax as is the case with Feederbot, a bot using DNS C&C we discovered during this work. Additionally, we discuss general architectural issues and limitations of DNS botnet C&C. The second group of related work contains approaches to detect botnets in network traffic. This kind of related work can be separated into application protocol specific approaches and protocol independent approaches. As HTTP is used as botnet C&C, some work has been done to develop specifically tailored detection methods for HTTP-based botnet C&C, such as Perdisci et al. [2]. Goebel and Holz [1] present methods in order to detect IRC-based botnets. However, due to their protocol-dependent orientation, none of these approaches are able to detect DNS C&C.

Independent of the application layer protocol, Gu [12] and Strayer [13] propose botnet detection methods based on network flow characteristics. However, protocol-independent approaches will likely fail to detect DNS C&C as they expose neither chat-style characteristics nor necessarily spatial-temporal correlated behavior. For example, Feederbot does neither exhibit periodicity nor synchronized transactions among different bot executions – effectively exploiting the gap of existing detection approaches. Therefore, we provide a detection method specifically tailored to DNS C&C based on rdata and behavioral communication features – successfully filling this gap.

A special case is the work of Choi et al. [14], which – though not specifically targeting DNS C&C – addresses group activities in DNS. The authors define differences between DNS query behavior typical to any kind of botnet and legitimate DNS resolution. According to Choi et al., key features for bot-typical behavior include simultaneous DNS queries, quickly changing C&C server addresses as well as transient domains. However, as our analysis of Feederbot discovers, none of these assumptions hold. Feederbot's C&C servers stayed up for the whole monitoring period of nine months and DNS queries are not synchronized between different bots. Instead, we exploit rdata features and persistent communication behavior to detect DNS C&C.

The third group of related work covers DNS covert communication. Bernat [15] analyzed DNS as covert storage and communication medium. Born and Gustafson [16] employ character frequency analysis in order to detect DNS tunnels. However, both approaches do not specifically address the detectability of DNS as botnet C&C. In addition, we significantly improve entropy-based features and combine them with behavioral features to target botnets.

VI. Conclusion

Inspired by anomalous DNS behavior, we stepped into a whole new kind of botnet C&C. This shows that even though many bot families use IRC or HTTP C&C, malware authors still find new ways of instructing their bots. It is obvious that DNS C&C moves botnet C&C one step further into the direction of covert communication. The detection of such botnet C&C even when covert, remains possible.

We combine protocol-aware information theoretical features with aggregated behavioral communication features and apply them at different levels of network traffic abstraction, i.e. DNS transactions and hosts. In this way, we detect DNS C&C in real-world DNS traffic. Furthermore, we provide means to classify malware concerning DNS C&C usage based on network traffic.

To summarize, to the best of our knowledge we are the first to not only describe a real-world botnet using DNS C&C, but also provide a mechanism to detect DNS C&C in network traffic.

Acknowledgements

We thank the anonymous reviewers and our shepherd Dieter Gollmann for many helpful comments. This work was supported by the Federal Ministry of Education and Research (Grant 01BY1110, MoBE).

References

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

[2] R. Perdisci, W. Lee, and N. Feamster, "Behavioral Clustering of HTTP-Based Malware and Signature Generation Using Malicious Network Traces," in *NSDI*, 2010.

[3] B. Stock, M. Engelberth, F. C. Freiling, and T. Holz, "Walowdac Analysis of a Peer-to-Peer Botnet," in *EC2ND*, 2009.

[4] 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*, 2009.

[5] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling, "Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm," in *USENIX LEET*, 2008.

[6] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A View on Current Malware Behaviors," in *USENIX LEET*, 2009.

[7] K. Chiang and L. Lloyd, "A Case Study of the Rustock Rootkit and Spam Bot," in *HotBots*, 2007.

[8] G. Gu, V. Yegneswaran, P. A. Porras, J. Stoll, and W. Lee, "Active botnet probing to identify obscure command and control channels," in *ACSAC*. IEEE Computer Society, 2009.

[9] S. Bromberger, "DNS as a Covert Channel Within Protected Networks," http://www.oe.energy.gov/DocumentsandMedia/DNS_Exfiltration_2011-01-01_v1.1.pdf.

[10] C. Rossow, C. J. Dietrich, H. Bos, L. Cavallaro, M. van Steen, F. C. Freiling, and N. Pohlmann, "Sandnet: Network traffic analysis of malicious software," in *Building Analysis Datasets and Gathering Experience Returns for Security*, 2011.

[11] C. R. D. Joan Calvet and P.-M. Bureau, "Malware Authors Don't Learn and That's Good!" in *MALWARE*, 2009.

[12] 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.

[13] W. T. Strayer, D. E. Lapsley, R. Walsh, and C. Livadas, "Botnet detection based on network behavior," in *Botnet Detection*, ser. Advances in Information Security, W. Lee, C. Wang, and D. Dagon, Eds. Springer, 2008, vol. 36, pp. 1–24.

[14] H. Choi, H. Lee, H. Lee, and H. Kim, "Botnet detection by monitoring group activities in DNS traffic," in *CIT*, 2007.

[15] D. Bernát, "Domain name system as a memory and communication medium," in *SOFSEM*, 2008.

[16] K. Born and D. Gustafson, "Detecting DNS Tunnels Using Character Frequency Analysis," http://arxiv.org/ftp/arxiv/papers/1004/1004.4358.pdf.