CoCoSpot: Clustering and Recognizing Botnet Command and Control Channels using Traffic Analysis[1]

Christian J. Dietrich[a,c], Christian Rossow[a,b], Norbert Pohlmann[a]

[a] *Institute for Internet Security, University of Applied Sciences Gelsenkirchen, Neidenburger Str. 43, 45877 Gelsenkirchen, Germany*
[b] *VU University Amsterdam, The Network Institute, The Netherlands*
[c] *Department of Computer Science, Friedrich-Alexander University, Erlangen, Germany*

Abstract

We present CoCoSpot, a novel approach to recognize botnet command and control channels solely based on traffic analysis features, namely carrier protocol distinction, message length sequences and encoding differences. Thus, CoCoSpot can deal with obfuscated and encrypted C&C protocols and complements current methods to fingerprint and recognize botnet C&C channels. Using average-linkage hierarchical clustering of labeled C&C flows, we show that for more than 20 recent botnets and over 87,000 C&C flows, CoCoSpot can recognize more than 88% of the C&C flows at a false positive rate below 0.1%.

Keywords: botnet C&C, botnet detection, traffic analysis, network security

1. Introduction and Problem Statement

A defining characteristic of a bot is its ability to be remote-controlled by way of command and control (C&C). Typically, a bot receives commands from its master, performs tasks and reports back on the execution results. All communication between a C&C server and a bot is performed using a specific C&C protocol over a certain C&C channel. Consequently, in order to instruct and control their bots, bot masters - knowingly or not - have to define and use a certain command and control protocol. The C&C protocol is thus considered a bot-inherent property.

Historically, bots used cleartext C&C protocols, such as plaintext messages transmitted using IRC or HTTP. However, a C&C channel relying on a plaintext protocol can be detected reliably. Methods such as payload byte signatures as shown by Rieck et al. [20] or heuristics on common C&C message elements such as IRC nicknames as proposed by Goebel and Holz [11] are examples for such detection techniques. To evade payload-based detection, botnets have evolved and often employ C&C protocols with obfuscated or encrypted messages as is the case with Waledac [7], Zeus [6], Hlux [27], TDSS [12], Virut [21] and Feederbot [8], to name but a

[1] The published version of this manuscript is available at `http://www.sciencedirect.com/science/article/pii/S1389128612002472` or `http://dx.doi.org/10.1016/j.comnet.2012.06.019`

Email addresses: `dietrich@internet-sicherheit.de` (Christian J. Dietrich), `rossow@internet-sicherheit.de` (Christian Rossow), `pohlmann@internet-sicherheit.de` (Norbert Pohlmann)

few. The change towards encrypted or obfuscated C&C messages effectively prevents detection approaches that rely on plaintext C&C message contents.

In this article, we take a different approach to recognize C&C channels of botnets and fingerprint botnet C&C channels based on traffic analysis properties. The rationale behind our methodology is that for a variety of botnets, characteristics of their C&C protocol manifest in the C&C communication behavior. For this reason, our recognition approach is solely based on traffic analysis.

As an example, consider a C&C protocol that defines a specific handshake – e.g., for mutual authentication – to be performed in the beginning of each C&C connection. Each request and response exchanged during this handshake procedure conforms to a predefined structure and length, which in turn leads to a characteristic sequence of message lengths. In fact, we found that in the context of botnet C&C, the sequence of message lengths is a well-working example for traffic analysis features. Table 1 shows the sequence of the first 8 messages in four Virut C&C flows and two Palevo[2] C&C flows. Whereas Virut exhibits similar message lengths for the first message (in the range 60-69) and a typical sequence of message lengths at positions five to eight, for Palevo, the first three message lengths provide a characteristic fingerprint.

ID	**Family**	**Message length sequence**							
		1	2	3	4	5	6	7	8
1	Virut	60	328	12	132	9	10	9	10
2	Virut	69	248	69	10	9	10	9	10
3	Virut	68	588	9	10	9	10	9	10
4	Virut	67	260	9	10	9	10	9	10
5	Palevo	21	21	30	197	32	10	23	10
6	Palevo	21	21	30	283	21	10	23	10

Table 1: Examples of message length sequences for Virut and Palevo C&C flows

Leveraging statistical protocol analysis and hierarchical clustering analysis, we develop CoCoSpot[3] , a method to group similar botnet C&C channels and derive fingerprints of C&C channels based on the message length sequence, the underlying carrier protocol and encoding properties. Furthermore, we design a classifier that is able to recognize known C&C channels in network traffic of contained malware execution environments, such as Sandnet [22].

The ability to recognize botnet C&C channels serves several purposes. A bot(net)'s C&C channel is a botnet's weakest link [10]. Disrupting the C&C channel renders a bot(net) ineffective. Thus, it is of high interest to develop methods that can reliably recognize botnet C&C channels. Furthermore, driven by insights of our analysis of botnet network traffic, we found that a bot's command and control protocol serves as a fingerprint for a whole bot family. Whereas for example properties of the PE binary change due to polymorphism, we witness that the C&C protocol and the corresponding communication behavior seldom undergo substantial modifications throughout the lifetime of a botnet. From an analyst's perspective, our classifier helps to detect and aggregate similar C&C channels, reducing the amount of manually inspected traffic.

To summarize, our main contributions are two-fold:

- We provide a clustering method to analyze relationships between botnet C&C flows.

[2]A synonym for the malware family Palevo is Rimecud (Microsoft terminology).

[3]CoCoSpot is derived from spotting Command and Control

- We present CoCoSpot, a novel approach to recognize botnet command and control channels solely based on traffic analysis features, namely carrier protocol distinction, message length sequences and encoding differences.

The remainder of this article is structured as follows. Section 2 sheds light on related work, defines the scope of this article and highlights innovative aspects of our approach. Subsequently, in Section 3, the general methodology as well as the feature space is described. While Section 4 deals with the clustering phase of C&C flows, Section 5 outlines the classifier which is then used to classify unknown flows. In order to evaluate our approach, as described in Section 6, we classified arbitrary network flows emitted from the dynamic malware analysis environment Sandnet as either C&C or Non-C&C and verified the results using two datasets. Finally, we discuss limitations of our approach in Section 7 and conclude in Section 8.

2. Related Work

Traditionally, botnet C&C channels have mainly been identified in two ways. First, publicly available blacklists [1, 13, 19, 23] provide lists of known botnet servers by IP addresses or domain names. The drawback of blacklists is that properties like IP addresses or domains are volatile. Bot masters can and do change these often, rendering detection methods based blacklists infeasible. In addition, botnets that rely on a peer-to-peer C&C architecture exhibit quickly changing sets of rendez-vous points. Some bot masters design their bot's bootstrap process even more resilient by avoiding any static rendez-vous coordinates, e.g., by using domain generation algorithms where the current rendez-vous point is valid for a very limited time span such as a few hours. Second, botnet C&C channels can be detected by checking for characteristic payload substrings. For example, Botzilla [20], Rishi [11] and rules for the Snort IDS [25] identify C&C channels in network traffic using payload byte signatures for a small set of known botnets. However, most encrypted or obfuscated C&C protocols do not exhibit characteristic payload patterns and undermine existing payload byte signatures.

Consequently, the traditional techniques are unsatisfying and have motivated research for automated and more reliable processes. In that trail of research, BotMiner [14] and BotGrep [18] provide approaches to use traffic analysis in order to find botnet C&C channels. However, while BotMiner requires detectable so-called A-plane activity such as spam, DDoS or scanning, *CoCoSpot* does not require any a priori or accompanying malicious actions and aims at the *recognition* of C&C channels. For *CoCoSpot*, in order to detect a bot, it is enough to just exhibit C&C communication. The graph-based approach of BotGrep requires botnets with distributed C&C architectures in order to detect them. However, *CoCoSpot* not only works with P2P botnets, but also with botnets exhibiting a centralized C&C architecture.

Jacob et al. present JACKSTRAWS [16], which exploits that certain C&C channels show recognizable system-level behavior in terms of system call traces of Anubis [5]. Particularly, JACKSTRAWS dynamically analyzes malware binaries (e.g., with Anubis) and models system call graphs for known C&C connections. New C&C channels are detected by matching unknown network connections against these graphs. As opposed to JACKSTRAWS, *CoCoSpot* does not depend on host-level analyses such as tainting, which enables our system to be applied without host access. In addition, we will show that *CoCoSpot* can detect C&C flows of numerous different bot families, while Jacob et al. do not provide insight into the diversity of the C&C channels detected by JACKSTRAWS. For example, while pure *download-and-execute*

C&C channels follow strict system call patterns, more complex C&C channels spanning multiple TCP/UDP connections (e.g., typical for most modern P2P- or HTTP-based botnets) may not be detected by JACKSTRAWS. In contrast, *CoCoSpot* detects different architectural and many semantical types of C&C channels.

Concurrent to our work, Bilge proposed DISCLOSURE [17], a system to detect C&C servers in unknown network traffic based on NetFlow data. Bilge frames a number of features that characterize botnet C&C servers, which are partially similar to ours, but are applied in a different context. While we use periodicity of messages to select C&C candidates, Bilge uses periodicity of network connections towards a server as an indicator for C&C channels. Similarly, we rely on the sequence of C&C message lengths, while DISCLOSURE bases on sequences of C&C stream size lengths. In both scenarios, these features seem to work well, while the constraints are different: DISCLOSURE is bound to the strict NetFlow format, while we can rely on much finer granularity (messages vs. connections) in our setting and deploy a wider flow format. Another advantage of *CoCoSpot* over DISCLOSURE is that we have fewer assumptions that narrow the type of C&C channels. For example, DISCLOSURE models network traffic in a client/server fashion and favors centralized C&C channel architectures and thus it will presumably fail for P2P-based botnets. Moreover, *CoCoSpot* does not require *a priori* knowledge on reputation of autonomous systems, so that *CoCoSpot* even recognizes C&C servers in well-reputable networks.

In general, our approach complements existing C&C detection systems, in that we propose a technique to also recognize the *nature* (i.e., malware family) of C&C channels. While the difference may sound subtle, we see a major contribution here. In many cases, it is desirable to know which type of botnet accounts for a detected C&C channel, e.g., to automate classification of malware [24]. In addition, especially compared to IP address (and domain) blacklists *CoCoSpot* provides a finer granularity in that it restricts the detection to specific flows, which turns out useful if legitimate and malicious activity appear on the same IP address (or domain). Moreover, our system is able to produce human-readable reports for detected C&C channels, making an analysis easy. Currently, *CoCoSpot* reports back the type of C&C channel found and provides the security analyst with examples of similar C&C channels in the training dataset.

In summary, the automatic recognition of C&C channels is a challenge and demands for a different approach. We fill this gap by designing a method to recognize known C&C channels based on traffic analysis while not relying on specific payload contents nor IP addresses or domain names.

3. Methodology

A coarse-grained overview of our methodology is shown in Figure 1. First, we dissect and aggregate TCP and UDP network traffic according to our network traffic data model. This process is described in Section 3.1. Based on this model, we design features that measure traffic analysis properties of network communication and extract these features from a set of manually analyzed C&C flows (Section 3.2). Using hierarchical clustering, we compile clusters of related C&C flows, and manually verify and label these C&C flows (Section 4). For each cluster, our method derives a centroid (Section 5.1) which is subsequently used during the classification of C&C candidate or even completely unknown flows of a contained execution environment such as Sandnet (Section 5.2).

Throughout the remainder of this work, we will use the following definition of the term *C&C protocol*.

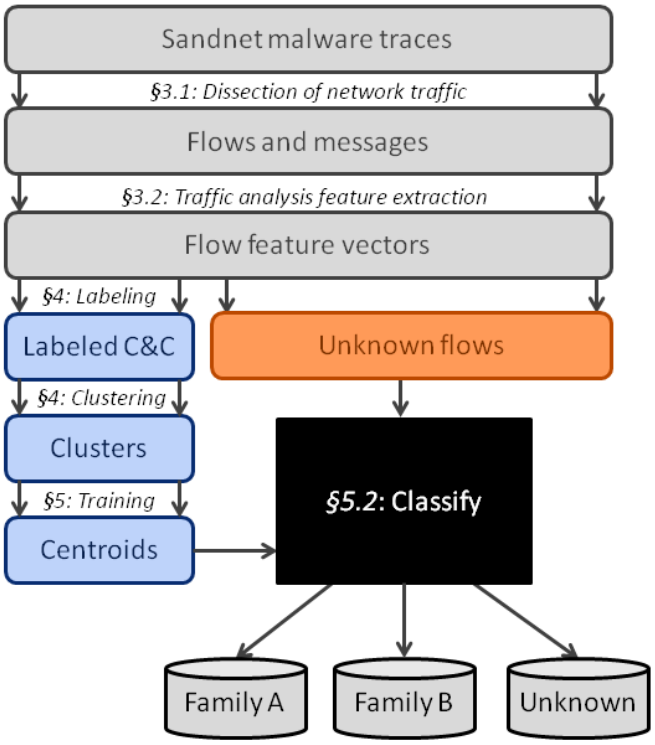

Figure 1: Overview of the C&C flow classification methodology

Definition 1. A C&C protocol is the protocol that is used to communicate instructions and reports, signal a bot's state and availability as well as transmit bot program updates between one or more bots and the controlling entity, the bot master.

3.1. Segmentation and Dissection of Network Traffic

In order to extract features from network traffic, it is essential to develop data structures which provide access to the syntactic fields of the involved network protocols. However, it is difficult to determine the level of detail required to achieve the goal of fingerprinting C&C channels in advance. On the one hand, a high resolution in the parsed data structures provides a fine-grained access to all fields of a specific network protocol. On the other hand, with performance and efficiency in mind, it is advantageous to parse only as few structures as required. We decided to reassemble TCP and UDP streams, parse HTTP and develop heuristics for the segmentation of unknown application layer protocols into messages. We designed a data model for TCP- and UDP-based network traffic and its dissected protocol information, providing two layers of abstraction, namely *flows* and *messages*.

A *flow* represents the notion of a communication channel between two entities in terms of one or more network connections. It is uniquely identified by the 4-tuple: transport layer protocol *l4p* (TCP or UDP), source s_{ip} and destination IP address d_{ip} as well as destination port d_p. We do not include the source port in the unique identifier of a flow, because in this context, we want a flow to be able to span multiple streams to the same destination IP address and port with different source ports. As an example, several HTTP connections between the same source and destination can have varying source ports whereas the transport layer protocol, the source and destination addresses as well as the destination port stay the same. Additionally, a flow contains contextual information such as start t_s and end times t_e and comprises the list of computed flow features, such as the number of messages and the results of payload-based protocol determination. Formally, we define a flow f

$$f := \langle\, l4p, s_{ip}, d_{ip}, d_p, t_s, t_e \, \langle features \rangle \,\rangle$$

Inspired by the fact that lots of application layer network protocols are designed in a dialogue-like fashion, e.g., pairs of request and response such as HTTP, SMTP and DNS, we heuristically split the packet payload of a flow into *messages* as follows. We define a message to be composed of all consecutive flow payload transmitted in one direction until the direction of the payload transmission changes, an inactivity timeout of 5 minutes is hit or a new stream is opened. The 5-minute-timeout stems from the fact that the network egress router of the contained environment has a stream idle timeout of 5 minutes. Frames without payload, especially those carrying only signalling information such as TCP acknowledgements are ignored because they do not contribute to the message payload. Additionally, we encountered flows which do not follow the request-response rhythm, mostly because several requests are sent before a response is received. In such a case, we treat multiple subsequent messages going in the same direction as duplicates and ignore them.

Formally, a message m_f of the flow f is thus defined as

$$m_f := \langle\, dir, len, t_s, t_e, payload \,\rangle$$

where *dir* denotes the direction in which the message was transmitted, e.g., source to destination or vice versa, *len* is the length of the message in bytes, t_s and t_e are timestamps of the message

start and end, and *payload* comprises the message's payload. In case of HTTP, a message is extended by dissected protocol-specific fields, namely the request URI and the request and response bodies. We use our custom HTTP parser to extract these fields for all streams recognized as HTTP by OpenDPI [15].

3.2. Traffic Analysis Features of Botnet C&C

Based on our data model, we define features aiming to measure traffic analysis properties of network communication, especially in the context of botnets. These features will be used for the clustering analysis and the subsequent classification of flows.

Our feature space consists of three features. First, we consider the carrier protocol to be the underlying protocol of the C&C protocol and distinguish between TCP, UDP and HTTP. We assume that a C&C protocol is either designed to be used with TCP, UDP or HTTP. In fact, we have not witnessed a change of carrier protocol during the lifetime of a botnet in our analysis of botnet traffic.

As the second feature, we define the sequence of message lengths in a flow. This feature is motivated by the observation that most C&C protocols exhibit a characteristic sequence of message lengths. More precisely, we exploit the characteristic message length sequence during the beginning of a flow. We assume that the first few messages of a flow cover what could be regarded a handshake phase of the C&C protocol. In addition, we observed that C&C flows exhibit characteristic message lengths during idle phases, i.e., when only keep-alive-like messages are transmitted. We decide to take up to the first eight messages into consideration because on the one hand they cover the initial handshake and – if there is an early idle phase – the first few messages of an idle phase, too. On the other hand, eight messages is reasonably small in order to keep the computational overhead low. We evaluated different numbers of messages in a flow and while smaller numbers make the performance decrease, we did not experience huge improvement with longer message length sequences.

In case of TCP and UDP as carrier protocol, the length of a message is defined as the total length of the carrier protocol's payload in bytes. For HTTP, we define the message length of an HTTP request to be the sum of the body length as well as the URI query section, and for a response the body length. We omit HTTP headers because they do not contain relevant information for our method.

The third feature is specific to HTTP and counts the number of distinct byte values in the query section of the HTTP request URI, aggregated over up to the first four HTTP requests' URIs of a flow. We observed that malware authors decide for different encoding schemes such as Base64, Hex or ASCII when designing an HTTP-based C&C protocol. To a certain degree, the number of distinct bytes reflect the encoding scheme of the URI's query section. For example, for a Base64 encoded query section of the URI, the number of distinct bytes does not exceed 64. Note that we restrict ourselves to the query section of the request URIs for all requests.

Throughout one botnet, it is possible that there are multiple C&C protocols with possibly even different C&C architectures in place, for several reasons. Malware authors might allow for a backup C&C protocol that is only activated if the primary C&C protocol fails, several variants of one bot family might operate in parallel using two distinct C&C protocols or the bot has been designed to work with two (or more) C&C channels. As an example, we observed the latter case with New_bb, an egg downloaded by Renos/Artro which exhibits two HTTP C&C channels with two different servers. Another example is Virut where some variants exhibit a plaintext IRC-based C&C channel while other more well-known variants use a TCP-based custom-encrypted C&C protocol.

4. Clustering Analysis of Known C&C Flows

Clustering enables us to identify and aggregate groups of similar C&C flows in our data set. We use clustering for two main reasons. Often, the message lengths of the messages in a C&C flow are not equally static throughout several C&C flows of one botnet, but show slight deviations in a small range. Thus, we need to aggregate similar flows and learn the range of each message's length in a C&C flow. Second, grouping similar C&C flows into clusters allows us to build a centroid for each cluster which represents the fingerprint of this cluster's C&C flows. The clustering step produces efficient representations that serve as training data for the subsequent classification of flows. In addition, the clustering results provide insights into and measure the relationships between clusters of different malware families.

4.1. Definition of the Distance Function

Using the features described in Section 3.2, we define the feature vector $v(f)$ of a flow f, called *flow vector*, as:

$$v(f) \;=\; \langle\; p, ml_1, ml_2, ml_3, ..., ml_n,\; hb \;\rangle$$

where $v.p$ denotes the carrier protocol TCP, UDP or HTTP, $v.ml_k$ denotes the length of the k-th message in the flow f, and hb is the number of distinct bytes in the query section of an HTTP request URI if the flow has HTTP as carrier protocol. We use $n = 8$, i.e., up to the first eight messages per flow, as described in Section 3.2. Based on the resulting feature space, we define the following distance function $d(u, v)$ of the feature vectors u and v of two C&C flows:

$$d(u,v) \;=\; \frac{1}{T}\, d_p(u,v) \;+\; \frac{1}{T}\, d_{ml}(u,v) \;+\; \frac{1}{T}\, d_{hb}(u,v) \tag{1}$$

where

$$T \;=\; \begin{cases} 3, & u.p = \texttt{http} \wedge v.p = \texttt{http} \\ 2, & \text{else} \end{cases} \tag{2}$$

$$d_p(u,v) \;=\; \begin{cases} 0, & u.p = v.p \\ 1, & \text{else} \end{cases} \tag{3}$$

and with k being the minimum number of messages in the two flow vectors u and v:

$$d_{ml}(u,v) \;=\; \frac{1}{k} \sum_{i=1}^{k} \left(\frac{|u.ml_i - v.ml_i|}{\max(u.ml_i, v.ml_i)} \right) \tag{4}$$

$$d_{hb}(u,v) \;=\; \begin{cases} \frac{|u.hb - v.hb|}{\max(u.hb, v.hb)} & u.p = \texttt{http} \wedge v.p = \texttt{http} \\ 0, & \text{else} \end{cases} \tag{5}$$

In the distance function d, all feature distance terms d_p, d_{ml} and d_{hb} weigh equally. If both flows are HTTP flows, then the three features p, ml and hb are each weighted with $1/3$, otherwise the two features p and ml are each weighted with $1/2$. The main intention of introducing weights in Equation 1 is to limit the range of output values to $[0, 1]$. While in general, weights can also be used to fine-tune the distance computation, we decide to keep the equal weights on purpose. Fine-tuning requires a representative evaluation dataset and if applied aggressively, fine-tuning inevitably leads to overfitting. In our case, using broad evaluation datasets, we will show that using the distance function with equally weighted feature terms yields very low misclassification

rates. When dealing with a very specific application or dataset, fine-tuning the weights might lead to a performance increase.

By definition, our distance function results in values between 0.0 (equal flows) and 1.0 (completely different flows). Table 2 consists of four flow vectors of the Virut family and two Palevo flow vectors and will be used to illustrate the distance computation. All Virut C&C flows have TCP as carrier protocol, Palevo flows have UDP as carrier protocol. The distance between the first two Virut flow vectors in Table 2 (IDs 1 and 2) is 0.0885. When looking at the first Virut flow vector (ID 1) and the first Palevo flow vector (ID 5), their distance is 0.4934.

ID	**Family**	**Carrier protocol**	**Message length sequence**							
			1	2	3	4	5	6	7	8
1	Virut	TCP	60	328	12	132	9	10	9	10
2	Virut	TCP	69	248	69	10	9	10	9	10
3	Virut	TCP	68	588	9	10	9	10	9	10
4	Virut	TCP	67	260	9	10	9	10	9	10
5	Palevo	UDP	21	21	30	197	32	10	23	10
6	Palevo	UDP	21	21	30	283	21	10	23	10

Table 2: Message length sequences for Virut and Palevo C&C flows (Table 1 extended with Carrier Protocol)

Even from this small subset of C&C flow vectors, it becomes obvious that the message length at a certain position is more characteristic than others. In our case, the messages at message position 2 of the Virut flows have varying lengths between 248 and 588 bytes whereas the messages at the first position vary in length between 60 and 69. We will take this into consideration when computing the cluster centroids and explain this in detail in Section 5.1.

4.2. Dataset Overview

We compiled several datasets in order to apply and verify our C&C flow recognition methodology. As a first step, aiming to get C&C candidate flows, we apply several heuristics to a set of 34,258,534 Sandnet flows, F_{all}, and consult the Amada C&C server blacklist in order to find C&C flow candidates. The flows in F_{all} are gained from Sandnet, a contained malware execution environment proposed by Rossow et al. [22], and stem from 34,387 malware binaries of 1590 distinct families according to Kaspersky antivirus labels. We conducted all malware execution experiments using a Windows XP SP3 32bit virtual machine connected to the Internet via NAT. We deploy containment policies that redirect harmful traffic (e.g., spam, infections) to local honeypots. We further limit the number of concurrent connections and the network bandwidth to mitigate DoS activities. An in-path honeywall NIDS watched for security breaches during our experiments. Other protocols (e.g., IRC, DNS or HTTP) were allowed to enable C&C communication. The biases affecting the following experiments due to containment should thus remain limited. We did not deploy user interaction during our experiments. The flows cover a time span from February 2010 to December 2011.

As heuristics to detect C&C flow candidates, we applied periodicity detection, long-lasting flow detection as well as domain flux detection to the flows in F_{all}. Note that while these heuristics certainly do not guarantee to find all C&C flows, we use them to find a bootstrapping set of C&C flows. The resulting set of candidate C&C flows is denoted as F_{Cand}. In order to detect periodic messages, we compute the time between any two subsequent messages in a flow, the so-called message gap interval in seconds, and the relative frequencies of these message gap

ID	# Flows	Description
F_{all}	34,258,534	Sandnet flows, mixed C&C and Non-C&C
F_{Cand}	23,162	C&C candidate flows
F_C	1,137	Manually verified C&C flows

Table 3: Data sets of flows

intervals. The message gap intervals are computed separately for requests and responses, and rounded to integer precision. A flow is considered periodic with period p if one message gap interval has a relative frequency of r_1 or two adjacent intervals add up to a cumulative relative frequency of r_2, in at least one direction. We evaluated r_1 between 45% and 60%, r_2 between 75% and 90% at a step size of 5, and finally chose $r_1 = 50\%$ and $r_2 = 80\%$.

We considered a flow as long-lasting if its duration is greater than half of the time that a malware sample was running in Sandnet. In addition, we heuristically detect a sample that performs a domain generation algorithm (DGA) by looking at the ratio of distinct DNS queries that result in NXDOMAIN to successful DNS responses for a sliding window time span of m minutes. Once the ratio exceeds g – i.e., more than g DNS queries result in NXDOMAIN in m minutes – we consider the presence of a DGA. In our case, we set $g = 100$ and $m = 5$. Furthermore, we used flows to IP addresses and domains of the Amada blocklist [1] as C&C candidates. In addition, we were provided with network traces of recently active botnets including Hlux [27] and Miner [28] with known C&C flows. We added these known C&C flows to the set F_{Cand}.

Of the C&C candidate flows, we manually reviewed and classified 2,691 flows as C&C and – if possible – assigned malware family labels. The resulting set of verified C&C flows is denoted as F_C.

At this stage, F_C contains 2,691 C&C flows of 43 bot families. This set is skewed, i.e., a few bot families cause the majority of C&C flows. We mitigate the imbalance in the set F_C by limiting the number of flows per family to a maximum of 50 and require a minimum of 10 flows per family. Finally, the filtered set contains 1,137 C&C flows of 43 distinct families, including e.g., Zeus, SpyEye, Sality P2P, Mariposa, Virut, Palevo/Rimecud, Hlux/Kelihos, Torpig, Koobface, Miner, Sirefef/ZeroAccess and Renos/Artro.

4.3. Hierarchical Clustering

We apply agglomerative hierarchical clustering to group the set of manually verified C&C channels F_C. In order to avoid the so-called chaining phenomenon, where the minimum distance between any two instances of adjacent clusters could cause the clusters to be merged, we decided to use average linkage hierarchical clustering, as the latter does not tend to fall for the chaining phenomenon. The clustering is performed 2-fold using two disjunct subsets to avoid overfitting, i.e., we split the dataset F_C into two halves, and cluster once using each half. Eventually, to group C&C flow vectors, a cut-off threshold determines the maximum distance for which two different flows still belong to the same C&C group (i.e., cluster). To illustrate, we again refer to the first two flow vectors in Table 2 (IDs 1 and 2) and their distance of 0.0885. If the cut-off threshold was greater than this distance, both instances would be aggregated in the same cluster. Otherwise they would be filed into different clusters. Finally, the most prevalent family per cluster in terms of number of flows determines the cluster's family. However, for some clusters we could not infer the malware family, because Antivirus scanners did not detect the samples at all, only heuristic or generic labels applied or labeling was inconsistent among Antivirus scanners. In these cases,

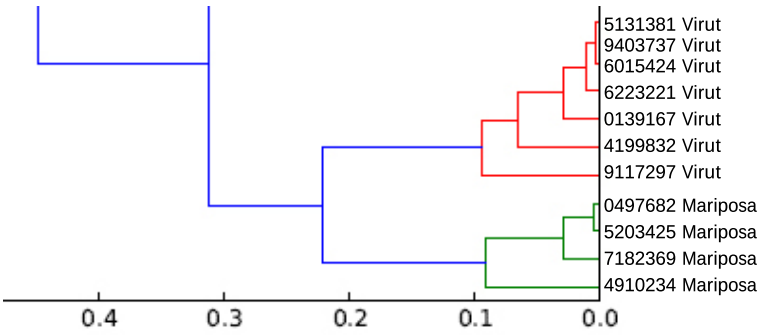

Figure 2: Example extract of a dendrogram that visualizes the clustering results, covering one Virut cluster and one Mariposa cluster and a cut-off threshold of 0.115.

we tried to manually assign a malware family based on the Antivirus scanning results and the IP addresses and domains used during C&C communication. Figure 2 provides an example dendrogram that visualizes the clustering results for one Virut and one Mariposa cluster.

4.4. Cluster Evaluation

We evaluate our clustering by checking if the clustering results correspond to the labels that we assigned to our training dataset. Ideally, for each family label (i.e., C&C protocol identifier) we assigned to the dataset, a cluster with no more than all elements with this label should be created. For that, in a first step, we have to choose a clustering cut-off threshold that results in the best-possible clustering result. To evaluate how well a clustering represents the labeled dataset, we use two measures from the area of unsupervised learning. First, we measure the *precision*, which represents how well our clustering separates C&C flows of different types. Second, we compute the *recall*, which expresses if similar C&C flows are grouped into the same cluster. Formally, let T be the set of clusters in the training dataset, C the set of created clusters, $m = |T|$, $n = |C|$ and s the total number of instances $s = \sum_{i=1}^{n} |C_i|$. We define precision P as

$$P = \frac{1}{s} \sum_{i=1}^{n} P_i = \frac{1}{s} \sum_{i=1}^{n} \max\left(|C_i \cap T_1|, |C_i \cap T_2|, ..., |C_i \cap T_m|\right) \tag{6}$$

and recall R as

$$R = \frac{1}{s} \sum_{i=1}^{m} R_i = \frac{1}{s} \sum_{i=1}^{m} \max(|C_1 \cap T_i|, |C_2 \cap T_i|, ..., |C_n \cap T_i|) \tag{7}$$

Note that we defined the overall precision and overall recall so that the precision and recall of all C&C types are equally taken into account. This mitigates imbalances that would otherwise have been introduced by datasets with non-equal numbers of elements per C&C type.

We aim for a clustering that both groups C&C flows of one type into one cluster (maximum recall), and that also separates between different C&C flows (maximum precision). At the later stage, a low precision translates to more false positives, while a low recall causes false negatives. In our setting, to avoid false positives, we can tolerate multiple clusters for one C&C type, but have to avoid too generic clusters by mixing different C&C types. We therefore combine

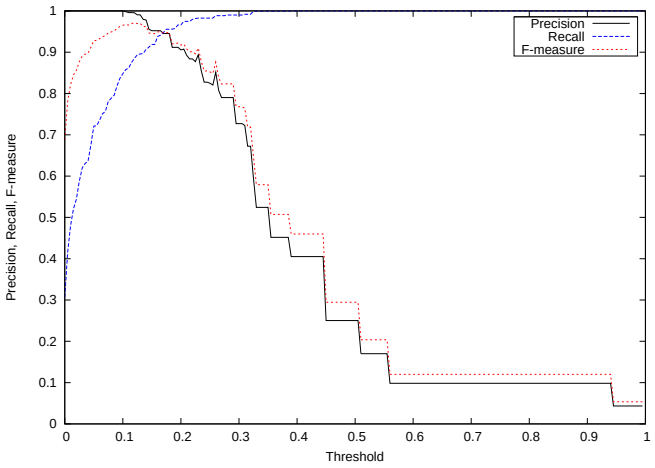

Figure 3: F-measure evaluation of the hierarchical clustering at different cut-off thresholds.

precision and recall in a weighted score and prioritize precision twice as important as recall. Formally, we use the *F-measure* [26] to evaluate the performance of a clustering with threshold th and $\beta = 1/2$:

$$\text{F-measure}_{th} = (1+\beta^2) \cdot \frac{P_{th} \cdot R_{th}}{\beta^2 \cdot P_{th} + R_{th}} \tag{8}$$

We then maximize this function to find the optimal clustering threshold th by iterating the cut-off-threshold in the range 0.0 to 1.0 using a step size of 0.005, as shown in Figure 3. The result of the F-measure evaluation leads to a cut-off threshold of 0.115.

4.5. Clustering results

The clustering phase results in a total number of 91 clusters, of which 13 are singletons, i.e., comprise only one C&C flow training instance. Table 5 provides an excerpt of the resulting clusters which includes those clusters that refer to well-known malware families. Some clusters could not reasonably be labeled with a malware family because the samples have not been detected by any Antivirus scanner at all (we used VirusTotal scan results), labeling was inconsistent, or only generic and heuristic labels applied. In total, the number of clusters outweighs the number of families in the training data set F_C. This is caused by families which have multiple clusters, such as Virut. We identified three main reasons for this. First, the bots of a certain malware family evolve over time. For Virut, two kinds of bots circulate in the wild. One variant is based on a plaintext IRC C&C channel, and another variant employs a TCP-based C&C channel which provides an intermediate layer of custom encryption. These two variants exhibit different message length sequences (and message encodings) which is reflected in two distinct Virut clusters. Second, different clusters might represent different activity states. Idle bots, i.e., where the bot master does not have instructions for the bots, typically exhibit smaller messages (e.g., only keep-alive messages) compared to when orders (e.g., spam templates and address lists) are transmitted. Third, families that do not exhibit characteristic message lengths at all will result in a high number of clusters.

5. Designing the C&C Flow Classifier

During the clustering phase, we have built groups of similar C&C flows. In order to extract fingerprints, we now compute a centroid for each cluster and design a nearest-cluster classifier that can classify unknown flows based on these centroids. Finally, we evaluate our classifier on two data sets and give upper bounds of classification errors.

5.1. Cluster Centroids as C&C Channel Fingerprints

In the training phase, we compute a centroid z for each cluster C.

The data structure z of a centroid is based on the structure of a flow vector v as defined in Section 4, extended by a sequence of weights and a maximum distance. It consists of the following attributes:

$$z = < p,\ ml_1, ml_2, \ldots, ml_n,\ w_1, w_2, \ldots, w_n,\ maxdist,\ hb >$$

- $z.p$: The carrier protocol.
- $z.ml_i$: The sequence of the average length of the messages at index i of all flows in C.
- $z.w_i$: The sequence of weights for each position i in the average message length sequence $z.ml$.
- $z.maxdist$: The maximum distance between the centroid and all of the flow vectors in the cluster in order to limit the cluster's scope.
- $z.hb$: In case of a centroid for a cluster with at least one HTTP C&C flow: the average number of distinct bytes in the query section of HTTP requests.

We compute the centroid z for each cluster C as follows. First, for each cluster, we load the output of the clustering step, i.e., the set of clustered C&C flow vectors of a cluster C. The carrier protocol of the centroid, $z.p$, is set to the carrier protocol of all flow vectors in C. Note that flow vectors with two different carrier protocols will never be clustered into the same cluster, as the clustering cut-off threshold is smaller than the distance caused by two differing carrier protocols. In other words, one cluster only spans flow vectors of one carrier protocol.

For all message length sequences of the flow vectors in cluster C, we compute $z.ml_i$ by averaging the elements at the i-th position in the message length sequence. Let C contain n flow vectors and let $V(C) = \{v_k\}_{k=1..n}$ be the set of flow vectors of the cluster C. For each flow vector v_k in $V(C)$ and each index i in the message length sequence of v_k, the centroid's sequence of message lengths is computed as follows:

$$z.ml_i(C) = \frac{1}{n} \sum_{k=1}^{n} v_k.ml_i \tag{9}$$

Referring to the example shown in Table 2, some message positions of the message sequences can be considered more characteristic for a C&C protocol due to less variation at a specific message position. In order to reflect this in the cluster centroid, we introduce a weighting vector which contains a weight for each message position and indicates the relevance of the message's position. The smaller the variation of the message lengths at a given message position of all flows in a cluster, the higher the relevance of this message position. In other words, if two flow

vectors' message lengths differ in a message position with low relevance, the less impact this has on the result of the classification distance function. Thus, we decide to compute the coefficient of variation (c_v) for each message position over all flow vectors in one cluster. The coefficient of variation [9] is defined as the ratio of the standard deviation to the mean and fits our needs. Consequently, we define our weight as one minus the coefficient of variation, in order to reflect that a higher variation leads to a smaller weight.

The weighting vector is computed as:

$$z.w_i(C) \;=\; 1 - \min\left(c_v(v_k.ml_i), 1\right) \;=\; 1 - \min\left(\frac{stddev(v_k.ml_i)}{mean(v_k.ml_i)}, 1\right) \tag{10}$$

ID	**Message length sequence**							
	1	2	3	4	5	6	7	8
1	301	2135	305	2169	305	2163	305	2153
2	301	2153	301	2149	305	2153	305	2123
3	301	2125	301	2153	305	2131	305	2157
4	301	2145	301	2155	305	2155	305	2115
	Average message length sequence							
	301	2139.5	302	2156.5	305	2150.5	305	2137
	Weighting sequence							
	1	0.995	0.994	0.997	1	0.994	1	0.991

Table 4: Examples for the average message lengths and weighting sequence of a centroid for a cluster of four C&C flows

Table 4 shows the flow vectors of a cluster with four C&C flows and the corresponding weights for all message positions. As shown, message positions with varying lengths have a weight value that decreases as the range of message lengths at that position increases.

In order to respect the weight in the distance computation during the classification of a flow vector, we modify the distance function for the message lengths d_{ml} in Equation 4 by adding the weight as a factor. The resulting distance function $d_{ml,class}$ is defined as:

$$d_{ml,class}(u, v) \;=\; \left(\sum_{i=1}^{k} z.w_i\right)^{-1} \cdot \sum_{i=1}^{k}\left(z.w_i \cdot \frac{|u.ml_i - v.ml_i|}{\max(u.ml_i, v.ml_i)}\right) \tag{11}$$

The complete distance function that is used during the classification is given as:

$$d_{class}(u, v) \;=\; \frac{1}{T}\, d_p(u, v) \;+\; \frac{1}{T}\, d_{ml,class}(u, v) \;+\; \frac{1}{T}\, d_{hb}(u, v) \tag{12}$$

where

$$T \;=\; \begin{cases} 3, & u.p = \texttt{http} \wedge v.p = \texttt{http} \\ 2, & \text{else} \end{cases} \tag{13}$$

In addition, we define the *quality indicator* of a cluster to be the normalized sum of all weights in the weighting vector. The quality indicator expresses the overall weight of all positions in a cluster centroid's message length sequence. As an example, if all of the message lengths in the message length sequence vary widely, this will result in a low quality indicator.

$$q(z) \;=\; \frac{1}{k} \cdot \left(\sum_{i=1}^{k} z.w_i\right) \tag{14}$$

The quality indicator is a means to filter clusters that do not represent characteristic message length sequences.

Family Label	*#C*	*#S*	*C&C Arch*	*CP*	*Plain*	*Avg QI*
Bifrose	4	1	centralized	TCP	no	90.48
BlackEnergy	3	0	centralized	HTTP	no	92.71
Cycbot/Gbot	6	0	centralized	HTTP	no	31.28
Delf	2	1	centralized	HTTP	no	95.33
Koobface	2	1	centralized	HTTP	no	88.89
Mariposa	6	1	centralized	UDP	no	86.41
Mebroot	6	2	centralized	HTTP	no	66.50
Miner	1	0	P2P	HTTP	yes	97.35
New BB	5	1	centralized	HTTP	yes	84.42
Nimnul/Ramnit	1	0	centralized	TCP	no	67.47
Palevo	2	1	centralized	UDP	no	86.28
Renos/Artro	2	0	centralized	HTTP	no	99.87
Renos/Katusha	1	0	centralized	HTTP	no	100.00
Sality P2P	6	2	P2P	UDP	no	74.31
Sirefef/ZeroAccess	3	0	P2P	TCP	no	80.51
Small	2	1	centralized	HTTP	no	44.50
Spatet/Llac	1	0	centralized	TCP	no	92.59
TDSS/Alureon	1	0	centralized	TCP	no	81.82
Tedroo	1	0	centralized	TCP	no	97.95
Torpig	5	0	centralized	TCP	no	71.55
Virut	3	0	centralized	TCP	mixed	82.01

Table 5: Clustering results of some well-known malware families. #C: number of clusters, #S: number of singleton clusters, C&C Arch denotes the C&C architecture (centr=centralized, P2P=peer to peer), CP is the Carrier Protocol, Plain denotes whether the family uses a plaintext C&C protocol encoding; Avg QI is the average quality indicator.

5.2. Classification Algorithm

So far, we have developed a data structure for the cluster centroid and a distance function for the classification. The complete algorithm to classify a flow f is as follows. First, we build the corresponding feature vector v_f and compute the distance between all cluster centroids and v_f using the distance function d_{class} in Equation 12.

Naive, we could be tempted to assign the closest cluster to the flow f, i.e., the cluster with the minimum distance to v_f. However, the closest cluster is not necessarily correct. Especially in case f is not a C&C flow at all, our classifier requires a means to find out that f lies outside the scope of all clusters. Thus, we additionally store the maximum distance between the centroid and all flow vectors of a corresponding cluster, computed using the distance function d of Equation 1. The maximum distance limits the scope of the cluster, and flows outside of this scope are discarded by the classifier.

The label of the nearest cluster, i.e., the cluster with the minimum distance between v_f and the cluster centroid, is assigned to f, as long as the distance is below the maximum distance for the centroid. If none of the cluster centroids are in range of f, the flow is considered not to be a known C&C flow.

ID	# Flows	Description
F_S	1,275,422	subset of Sandnet flows, only Non-C&C
F_{FN}	87,655	C&C flows to C&C peers

Table 6: Evaluation data sets

6. Evaluation of the C&C Flow Recognition

In this section we use our classifier to recognize C&C channels among arbitrary network traffic emitted from Sandnet, our contained malware execution environment. Due to the sheer amount of flows to be classified and the absence of rock-solid ground truth for C&C flows, validating the result of the classification is a difficult task. Therefore, we try to estimate upper bounds for the classification errors in order to provide an impression of the performance of our classification approach. We verify our classifier by help of two evaluation datasets, shown in Table 6.

In order to estimate the false negative rate, we compiled a set of C&C peers. For each of the 43 families in the training set F_C, we build sets of C&C server IP addresses or domain names based on manually verified C&C flows as well as publicly available C&C server blacklists [2–4]. We assume all flows heading for any of the IP addresses or domains in our list of C&C peers as C&C flows. Using these C&C peer lists, we extract all matching flows from Sandnet traffic F_{all}, except those in the training set F_C. For five families, our heuristic did not find additional C&C flows, i.e., all flows of that specific family have already been used in the training phase and are thus excluded from the classification set F_{FN}.

For an initial set of centroids for classification, we discard centroids of four C&C families, as these do not show sufficiently characteristic message lengths. Among these families are SpyEye, Hlux, Zeus P2P and Carberp. Particularly, we discard families with only singleton clusters, or centroids whose average message length weights is smaller than 30%. Our idea of finding a C&C protocol's handshake could be extended to search for representative message length sequences past our current limit of eight messages per flow to mitigate these cases. The resulting data set is denoted as F_{FN} and contains C&C flows for 34 out of 43 families.

In order to estimate the false positive rate, we build a subset of all Sandnet flows F_{all}, balanced by A/V label family so that the subset contains no more than five malware samples of a certain family, determined by Kaspersky antivirus labels. The resulting set of flows to be classified is denoted as F_S, and contains a variety of different application-layer protocols from 3245 different malware samples of 671 distinct families. From this set, we remove all flows destined to any of the IP addresses or domains of the C&C peers, so that the resulting set does not contain any C&C flows of the families known to our classifier.

The classifier is trained on the clustering output of the data set F_C of Section 4.4. The training subset contains up to 50 random C&C flows per family. Note that we made sure that none of the flows used during the training phase of the classifier were in either of the sets of flows to be classified.

We define two types of classifications. First, if a C&C flow of F_{FN} actually belongs to family A (based on the C&C peer list), but our classifier assigned a different family (e.g family B) or no family at all, this is considered a *false negative*. Thus, if our classifier assigned the same family as assigned by the C&C peer list, we denote this as *true positive*. Per family, the true positive rate is the ratio of correctly assigned flows over all C&C flows for a specific family. Second, if a Non-C&C flow of F_S is assigned a C&C cluster, we denote this as *false positive.* The false

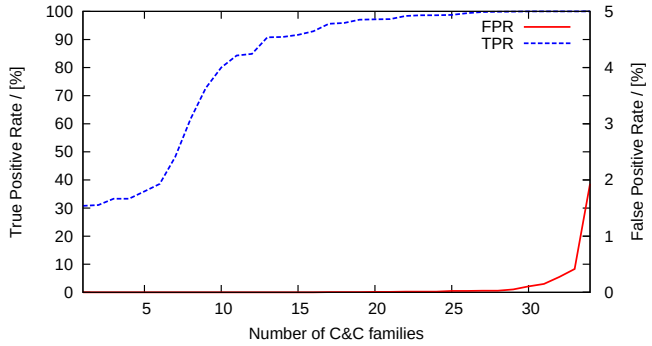

Figure 4: Overview of the C&C flow classification performance.

positive rate is the ratio of falsely classified flows for a cluster family over the total number of flows in F_S.

Figure 4 shows the results of the classification as a cumulative distribution function for all families that were included in both, true positive and false positive analysis. Note that the true positive rate values are given on the left y-axis, the false positive rate values on the right y-axis. More than half of all families have a true positive rate of over 95.6%. A small fraction of seven C&C families had true positives rates lower than 50%, which was caused by too specific cluster centroids. In most of these cases, we had too little training data to learn representative message length variations of a particular C&C protocol, which could be improved by adding more training data for this family.

For 88% of the families, the false positive rate is below 0.1%, and 23 cluster families do not exhibit any false positive at all. For the few families that cause false positives, we observed that the corresponding cluster centroids have high variation coefficients on many message length positions, effectively rendering the centroids being too generic and possibly matching random flow patterns. The C&C family Cycbot/Gbot shows a maximum false positive rate of 1.93%, while having a true positive rate of 99.92%. At the same time, the average quality indicator of the Cycbot/Gbot family is 31.28% which is the minimum of all families. The Cycbot/Gbot family exhibits an HTTP-based C&C protocol and after the clustering, it spreads among six clusters, which we observed due to several different activity states. Four of the six clusters are caused by C&C server errors, such as servers returning errors in the HTTP 500-599 status code range, File not found (status code 404) and even cases where servers returned OK (status code 200), but the response body contained an error message, telling that the requested page could not be found. In summary, erroneous C&C flows in the training data for this family lead to false positives where Non-C&C flows matched which happened to exhibit similar requests and had the same response bodies returned (e.g., for status code 404 File not found or 500 Internal server error). When compared to other families, the low quality indicator for Cycbot/Gbot shows that, on average, the Cycbot/Gbot clusters do not tightly represent characteristic behavior.

7. Discussion

Whereas the previous chapters presented a method to recognize C&C channels by help of clustering and subsequent classification, this section will shed light on the strengths of our

method as well as possible evasions. As is often the case, a detection approach such as ours can be mitigated by adversaries. However, we believe that today, only very few of the recent botnet C&C channels have been designed to mitigate the kind of traffic analysis proposed in this work. We provide answers to the question why our methodology works in recognizing C&C channels and we will outline the limitations of our approach.

Why does our methodology work? We assume that, driven by the practical need to evade payload byte signatures, botnet C&C protocols have evolved from plaintext to obfuscated and encrypted message encodings. However, we speculate that in practice, it has so far hardly ever been required to address the evasion of traffic analysis from a bot master's point of view.

The fact that our approach relies on message length sequences sets a limitation if messages would no longer exhibit characteristic lengths. During the clustering phase, we had to remove four families from the dataset because they do not have characteristic message lengths. These families include Hlux, SpyEye, Zeus P2P and Carberp. However, it is difficult to tell whether the C&C protocols of these families are designed to evade traffic analysis approaches or whether CoCoSpot just failed to derive centroids due to the kind of data that was transmitted over the labeled C&C flows. We manually inspected the families where our approach has failed, in order to clarify whether the message lengths are altered on purpose or by coincidence. Only in case of Zeus P2P, using reverse engineering, we found that it adds random message padding to most of its C&C messages before encryption, possibly to evade message-length-based approaches such as CoCoSpot. Interestingly, in case of Sality P2P, while message lengths vary, certain C&C messages are not padded at all and do exhibit characteristic message lengths. Thus, even if a subset of the C&C messages exhibits characteristic message lengths, CoCoSpot can effectively detect and classify this family. Furthermore, telling from the evaluation results, 34 of the 43 malware families we analyzed, can successfully be detected using CoCoSpot. We see our approach as an innovative, additional means of classifying malicious traffic.

In addition, our methodology has some limitations concerning details of the practical implementation. As described in Section 3.1, we heuristically split messages of a flow if the direction of transmission changes or a new connection is opened. If several messages are sent in the same direction over the same connection, i.e., the C&C protocol does not follow a dialogue-like pattern, our approach will fail to correctly separate these. Indeed, after manual inspection, we found such cases to cause false negatives, especially with unreliable C&C servers that did not respond to an unusually high number of requests. For example, if a C&C server only responds to every fifth request, five subsequent requests might be aggregated into one request message until a response is received. Thus, the resulting message length sequence differs significantly from what has been learned during the training phase (where the C&C server responded to every request), exhibiting higher values for the request message lengths. However, if the carrier protocol is known, e.g., in case of HTTP, it can be parsed and does not need to be split heuristically. Retransmissions of requests without response could then be ignored and messages can be separated at well-defined boundaries. This could further reduce the possible false classification.

Additionally, care has to be taken during the training phase in order to make sure that representative C&C flows are considered. As the classification results of the Cycbot/Gbot family show, erroneous C&C flows in the training data might result in high false positive rates.

8. Conclusion

With CoCoSpot, we have shown that for a variety of recent botnets, C&C protocols can be detected using traffic analysis features, namely the message length sequence of the first 8

messages of a flow, the carrier protocol as well as differences in the encoding scheme of the URI's query section in case of HTTP messages. The huge benefit of our approach is to be independent from payload byte signatures which enables the detection of C&C protocols with obfuscated and encrypted message contents. In addition, our C&C flow fingerprints complement existing detection approaches while allowing for finer granularity compared to IP address or domain blacklists. As a side-effect, our C&C flow clustering can be used to discover relationships between malware families, based on the distance of their C&C protocols. Experiments with more than 87,000 C&C flows as well as over 1.2 million Non-C&C flows have shown that our classification method can reliably detect C&C flows for a variety of recent botnets with very few false positives. Inspired by CoCoSpot, we plan to foster research in the area of botnet detection using traffic analysis.

Acknowledgements

We thank Felix C. Freiling and Christian Nordlohne for many helpful comments, Tillmann Werner for providing network traces of recent botnets, the guys at Anubis and the anonymous sample providers for providing us with samples. This work was supported by the German Federal Ministry of Education and Research (Grant 01BY1110, MoBE).

Vitae

Christian J. Dietrich

In 2008, Christian J. Dietrich finished his MSc in Computer Science at the University of Applied Sciences Gelsenkirchen. Since then, he is pursuing his PhD at the Friedrich-Alexander-University Erlangen, Germany. His research focusses on the network-based detection of malicious software. Christian J. Dietrich is one of the main developers of Sandnet, a large-scale contained environment to dynamically analyze malicious software.

Christian Rossow

Christian Rossow finished his MSc in Computer Sciences at the Vrije Universiteit Amsterdam in 2009 and continued with his PhD studies under Herbert Bos and Maarten van Steen. Since 2006, Christian is also a research assistant at the Institute for Internet Security if(is) in Gelsenkirchen. He is one of the main developers of Sandnet, a large-scale contained environment to dynamically analyze malicious software. Next to this, his research interests are exploring botnet detection techniques and analyzing the malware ecosystem.

Norbert Pohlmann

Norbert Pohlmann is professor for Distributed Systems and Information Security at the University of Applied Sciences Gelsenkirchen, Germany. In 2005, he founded the Institute for Internet Security if(is), a laboratory with focus in security research areas such as early warning systems, botnets, trusted computing, mobile security and identity management. He is also chairman at the TeleTrusT security society, steering committee member of the German task force for IT security, and adviser of eco, the largest German Internet economy society.

References

[1] abuse.ch, 2011. AMaDa Blocklist. http://amada.abuse.ch/blocklist.php, [Online; accessed 16-December-2011].

[2] abuse.ch, 2011. Palevo Tracker IP Address and Domain Blocklist. http://amada.abuse.ch/palevotracker.php, [Online; accessed 16-December-2011].

[3] abuse.ch, 2011. SpyEye Tracker IP Address and Domain Blocklist. https://spyeyetracker.abuse.ch/blocklist.php, [Online; accessed 16-December-2011].

[4] abuse.ch, 2011. Zeus Tracker IP Address and Domain Blocklist. https://zeustracker.abuse.ch/blocklist.php, [Online; accessed 16-December-2011].

[5] Bayer, U., Kruegel, C., Kirda, E., April 2006. TTAnalyze: A Tool for Analyzing Malware. In: 16th Annual EICAR Conference, Hamburg, Germany.

[6] Binsalleeh, H., Ormerod, T., Boukhtouta, A., Sinha, P., Youssef, A. M., Debbabi, M., Wang, L., 2010. On the analysis of the zeus botnet crimeware toolkit. In: PST. IEEE, pp. 31–38.

[7] Calvet, J., Davis, C. R., Bureau, P.-M., 2009. Malware authors don't learn, and that's good! In: 4th International Conference on Malicious and Unwanted Software (MALWARE), 2009.

[8] Dietrich, C. J., Rossow, C., Freiling, F. C., Bos, H., van Steen, M., Pohlmann, N., 2011. On Botnets that use DNS for Command and Control. In: Proceedings of European Conference on Computer Network Defense.

[9] Dodge, Y., 2006. The Oxford Dictionary of Statistical Terms.

[10] Freiling, F. C., Holz, T., Wicherski, G., 2005. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In: di Vimercati, S. D. C., Syverson, P. F., Gollmann, D. (Eds.), ESORICS. Vol. 3679 of Lecture Notes in Computer Science. Springer, pp. 319–335.

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

[12] Golovanov, S., Rusakov, V., 2010. TDSS. http://www.securelist.com/en/analysis/204792131/TDSS, [Online; accessed 16-December-2011].

[13] Google, 2011. Safe Browsing API. http://code.google.com/apis/safebrowsing/, [Online; accessed 16-December-2011].

[14] Gu, G., Perdisci, R., Zhang, J., Lee, W., August 2008. BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. In: 17th USENIX Security Symposium, San Jose, CA.

[15] ipoque.com, 2011. OpenDPI. http://www.opendpi.org/, [Online; accessed 16-December-2011].

[16] Jacob, G., Hund, R., Kruegel, C., Holz, T., 2011. JACKSTRAWS: Picking Command and Control Connections from Bot Traffic. In: USENIX Security Symposium 2011.

[17] Leyla Bilge, 2011. Network Based Botnet Detection. Ph.D. thesis, TELECOM ParisTech.

[18] Nagaraja, S., Mittal, P., Hong, C.-Y., Caesar, M., Borisov, N., 2010. Botgrep: Finding p2p bots with structured graph analysis. In: USENIX Security Symposium. USENIX Association, pp. 95–110.

[19] PhishTank, 2011. Phish Archive. http://www.phishtank.com/, [Online; accessed 16-December-2011].

[20] Rieck, K., Schwenk, G., Limmer, T., Holz, T., Laskov, P., 2010. Botzilla: Detecting the Phoning Home of Malicious Software. In: SAC 2010.

[21] Rossow, C., Dietrich, C. J., Bos, H., July 2012. Large-Scale Analysis of Malware Downloaders . In: Proceedings of the 9th Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA '12) . Heraklion, Greece.

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

[23] Spamhaus, 2011. XBL. http://www.spamhaus.org/xbl/, [Online; accessed 16-December-2011].

[24] Stone-Gross, B., Kruegel, C., Almeroth, K., Moser, A., Kirda, E., 2009. FIRE: FInding Rogue nEtworks. In: ACSAC.

[25] Threats, E., 2011. Snort Ruleset. http://www.emergingthreats.net/, [Online; accessed 16-December-2011].

[26] van Rijsbergen, C. J., 1979. Information Retrieval. Butterworths, London.

[27] Werner, T., 2011. Botnet Shutdown Success Story: How Kaspersky Lab Disabled the Hlux/Kelihos Botnet. http://goo.gl/SznzF, [Online; accessed 16-December-2011].

[28] Werner, T., 2011. The Miner Botnet: Bitcoin Mining Goes Peer-To-Peer. http://goo.gl/NSXnl, [Online; accessed 16-December-2011].