This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

Botnet Communication Patterns

Gernot Vormayr, Tanja Zseby, and Joachim Fabini

***Abstract*—Malicious botnets have become a common threat and pervade large parts of the Internet today. Existing surveys and taxonomies focus on botnet topologies, Command and Control (C&C) protocols, and botnet objectives. Building on these research results, network-based detection techniques have been proposed that are capable of detecting known botnets. Methods for botnet establishment and operation have evolved significantly over the past decade resulting in the need for detection methods that are capable of detecting new, previously unknown types of botnets.**

In this paper we present an in-depth analysis of all network communication aspects in botnet establishment and operation. We examine botnet topology, protocols, and analyze a large set of very different and highly sophisticated existing botnets from a network communication perspective. Based on our analysis, we introduce a novel taxonomy of generalized communication patterns for botnet communication using standardized Unified Modeling Language (UML) sequence diagrams. We furthermore examine data exchange options and investigate the influence of encryption and hiding techniques. Our generalized communication patterns provide a useful basis for the development of sophisticated network-based botnet detection mechanisms and can offer a key component for building protocol- and topology-independent network-based detectors.

***Index Terms*—bot, botnet, C&C, botnet detection**

I. Introduction

NETWORKED computers enable distributed computing and sharing of resources. Distributing tasks over multiple machines allows the execution of tasks whose demands exceed the resources available on one single computer. This technique can also speed up the processing of a single task by splitting it up into sub tasks that can be performed on several computers simultaneously.

One option to design such systems is by combining resources on networked computers, which are called bots, designed to perform a task or sub tasks.

An additional networked computer, called the master, is needed for coordinating the bots. Hence, the given name botnet, which is a combination of bot and network.

Since the Internet contains vast amounts of unused processing power and network bandwidth, malicious actors found ways to use those machines for their goals. This is achieved by infecting machines with malicious software (malware), and therefore building a malicious botnet, which uses compromised machines for computation.

Manuscript received ...

G. Vormayr, T. Zseby, and J. Fabini are with Institute of Telecommunications, TU Wien, Vienna, Austria (email: {first.last}@nt.tuwien.ac.at)

Digital Object Identifier ...

A. Malicious Botnets

Malicious botnets are used for various tasks including information theft [1], [2] or abuse of online services [3]. Another large percentage of botnets is employed for service disruption, which is used for stifling other players (e.g., competition, foreign states) [4], [5]. Those malicious botnets pose a real threat to individuals on the Internet [2], [6], networked infrastructures [7], and even the Internet overall [4], [8]. Therefore, various taxonomies [9]–[11] and recommendations [12] categorizing different aspects of botnets and ways of detection have been published. In line with the majority of earlier publications, the remainder of this work uses the term botnets to refer to malicious botnets.

A botnet consists of i) several bots, ii) a Command and Control (C&C) server, and iii) a botmaster. Additionally, this work uses the term victim for the target of an attack or a bot infection. The diagram in fig. 1 illustrates botnet participants, roles, and communication in a hypothetical botnet with centralized command structure.

Bots (also called zombies [13]) are infected machines that execute the bot executable. Those machines process the tasks supplied by the C&C server.

The C&C server is the central rallying point for the botnet. Since this server is capable of controlling every bot, it has to be protected from takeover by third-parties like, e.g., law enforcement, researchers, or rivaling bot masters. Modern botnet topologies (e.g., Peer to Peer (P2P)) can utilize regular bots that are part of the botnet as C&C server. Thus, the threat of bringing down the botnet via a single point of failure is remedied.

The botmaster is the person controlling the botnet via the C&C server. Botmasters try to stay as stealthy as possible, because unmasking the identity of the botmaster can lead to potential prosecution. This can be achieved for example by using as little traffic as possible, or by not using a direct connection to the rest of the botnet.

B. Botnet Detectors

Botnets taking part in possibly illegal activity can cause unwanted network traffic, or interfere with day to day business. Thus, detection and removal of botnets are important tasks. Since botnets tend to hide their operation, detection of botnets is an active field of research. Botnet detection can target one of the three parts of a botnet (bot, C&C server, or botmaster, as has been explained above and in fig. 1). Although this work concentrates on bot detection, the topologies described in section III and the protocols described in section IV also concern the C&C server. Furthermore, the C&C server is part of the presented communication patterns in section VI. A

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

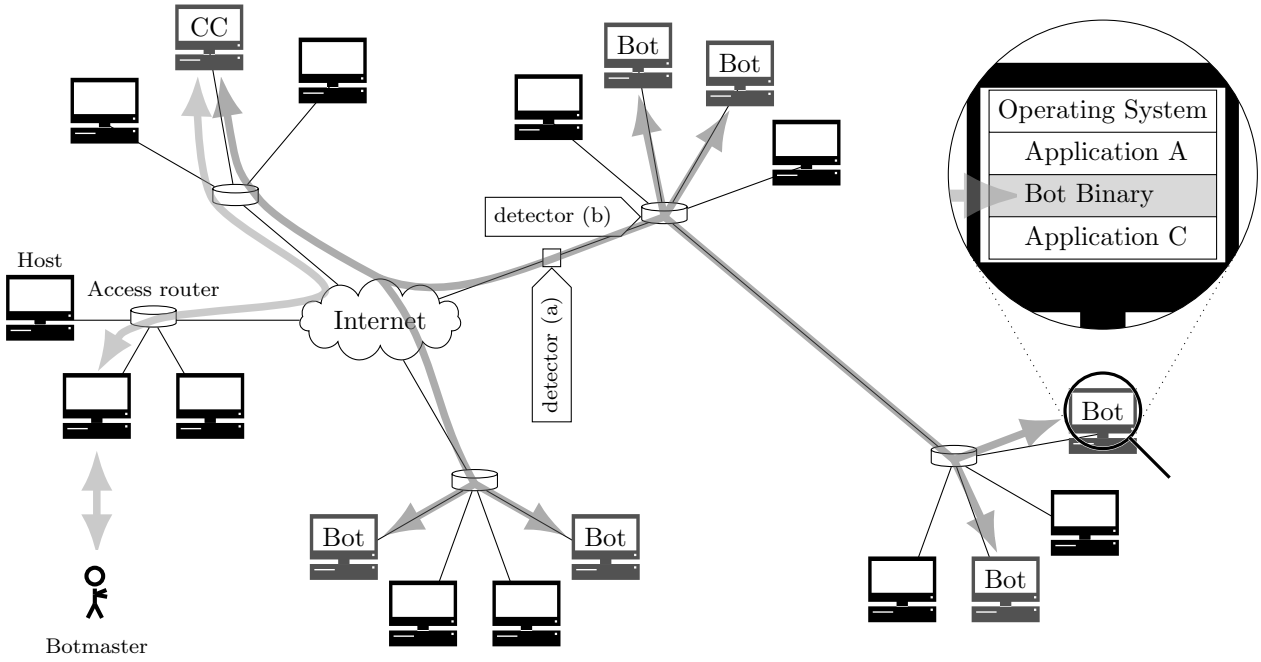


Fig. 1. Botnet overview example consisting of a centralized botnet including botmaster, C&C server, communication channels, bots, and bot binary. Additionally, example network-based detector positions (a) and (b) are depicted. Detector (a) acts as an intermediate node while (b) is included in a router.

communication pattern is a sequence of exchanged messages needed for a specific communication scenario. Thus, this work can also be used as a base for building a C&C server detector.

Since network communication is paramount for a botnet, it has to be present and can therefore be used for bot detection. This is called network-based bot detection. Example positions for network-based botnet detectors can be seen in fig. 1 (Detectors (a) and (b)).

The least complex approach for implementing a network-based detector is signature-based detection or syntactic detection [9]. This technique works by first collecting communication from known botnets. Next, representative byte-sequences or text excerpts used by the targeted botnets during C&C communication can be extracted from these samples. In the last step these results are compared with unknown traffic. If there is a match, bots have been detected.

Since the above mentioned byte-sequences or text excerpts need to match exactly, a straightforward countermeasure for hiding from such detection techniques is to modify the used C&C commands. A generalization of this idea is to add random sequences at the beginning or at arbitrary positions of a C&C command, which is called padding. Adding data at the beginning can be used to exploit length restrictions in detectors, while adding patterns at other locations can defeat the signature detection itself. Although padding and protocol variations can be mitigated by splitting up the captured communication into sub patterns, protocol encryption can thwart every signature-based detection.

Behavioral network analysis can be used to overcome this limitation. This technique relies on classifying network traffic as either normal traffic or anomalous traffic [9]. Therefore, this technique is also called network anomaly detection.

One way to build a network anomaly detector is to observe "abnormal" behavior (e.g., botnet C&C traffic) and to find characteristic patterns or properties (e.g., packet timing, packet count, packet length). These properties are called features. The same can be achieved by observing "normal" behavior (i.e., legitimate traffic captured during network operation), extracting features, and defining network traffic that does not match these patterns as an anomaly.

No inspection of the payload is needed, and therefore this approach works with encrypted or obfuscated network traffic. Since this technique tries to find generalized properties of unwanted traffic, it can theoretically be used to detect previously unknown botnets.

A simple way to implement this procedure is to use machine learning. Machine learning is a process which modifies the parameters of an algorithm until the algorithm output matches the expected output for given inputs. This is called training. A trained machine learning algorithm can be an approximation of the dependencies between the trained parameters and the expected results. Since there is not necessarily an underlying correlation in the data, the results have to be validated and used with caution.

Various detectors have been proposed that train different machine learning techniques with observed botnet traffic. Saad et al. [14] used the above mentioned approach of extracting features from network traffic, training a machine learning algorithm with known data, and using the trained algorithm for detection botnets. The algorithms were trained with normal and botnet traffic. For evaluation, this procedure was used on five different machine learning techniques. The results compare the different algorithms according to training and classification speed, and detection performance. Saad et al. [14] concluded that by using this simple approach it is possible to detect botnets with analyzing only the packet metadata.

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

However, those detectors are neither capable of adapting to changes in network traffic nor able to detect novel botnets.

Alauthaman et al. [15] extended this simple approach with adding a filtering step before the machine learning and adding an automated feature selection. This works by adding a machine learning algorithm that is used for discarding features which have only a small influence on the result. Better results in detecting known botnets were achieved than in previous works, but the performance on unknown botnets was not considered.

Adaptability was considered in the botnet detector design of Bilge et al. [16]. This botnet detector was also based on the simple approach mentioned above, but instead of detecting bots it was designed to detect C&C servers. To lower the number of wrong classifications of benign nodes as C&C servers, the result was combined with results from various blacklists. A blacklist is a list containing elements that are not allowed access. In the case of botnets, these lists contain addresses or hostnames of known C&C servers. Bilge et al. [16] tested the botnet detector in different sized networks and also tried it out on modified botnet communication. The results for variability in botnet communication look promising, but only variations in communication timing were tested, and therefore the performance for detecting unknown botnets was not tested.

The detectors mentioned above train the machine learning algorithms with normal traffic for the normal class and with observed botnet traffic for the anomalous class to achieve a high accuracy. This implies that only the botnets which were encountered during training and close relatives can be detected. Therefore, the above stated goal of detecting unknown botnets can not be achieved [17], [18].

The detectors in [19]–[21] try to overcome this limitation by considering typical botnet tasks during the traffic analysis (e.g., Distributed Denial of Service (DDoS) attacks, distributing unwanted electronic mails (emails), network scanning). A more detailed explanation of these works can be seen in section II. Since this additional information is used to train for specific tasks or protocols, the detection capability of unknown botnets is still limited [9], [18].

C. Structure of the Paper

In this paper we provide key components for building a network-based botnet detector. These key components are derived from a diverse list of existing botnets which were specifically chosen to include botnets differing in age, topology, used protocols, and complexity.

The remainder of this work starts with a discussion of related work in section II.

This is followed by a detailed introduction into botnet topologies in section III. Every topology and the possible variants are described based on existing botnets. The different variants are explained in detail including advantages, disadvantages, and historic context.

After the botnet topologies, a comprehensive list of botnet C&C and operational protocols in use is given in section IV. The protocols are illustrated and categorized according to possible topologies, challenges which arise from these combinations, and botnets using those protocols. Finally, hiding and obfuscation techniques are listed.

Section V gives a detailed overview of the botnets analyzed in this work.

In section VI we present our taxonomy of botnet communication patterns which cover typical usage scenarios of botnets. Those communication patterns are described in detail, including analysis of existing botnets in several subsections and visualization with Unified Modeling Language (UML) sequence diagrams. These sequence diagrams can be used as a base for building protocol and topology independent network-based botnet detectors which can even detect botnets using encryption or obfuscation techniques. Additionally, these diagrams fulfill every botnet use case, which allows the detection of previously unknown botnets.

Finally, section VII concludes the work with closing remarks.

II. Related Work

Botnets are a highly researched area. Many works have been published that analyze or categorize botnets and their behavior [9]–[11], [22], [23]. However, protocol and topology independent network communication patterns needed by botnets are not covered by those. Despite the lack of research in this area, some detectors [19]–[21] have been proposed that attempt to take network behavior into account.

Bailey et al. [22] have presented a survey of botnet propagation mechanisms and botnet topologies. This survey extends previous research with an analysis of attack classes. Additionally, a shift in propagation mechanism from operating system exploits to higher level applications (e.g., web browsers) and social engineering has been observed. This trend continues today with many botnets spreading only via spam and web sites. The survey concludes with proposed detection mechanisms based on correlation, signatures, or attack behavior.

The survey [23] by Eslahi et al. covers the same topics as Bailey et al. [22], but in more detail. Additionally, the bot life cycle consisting of infection, rallying, commands and reports, and abandon is introduced. Furthermore, future challenges like changing botnet techniques and environments, single bot detection, and Hyper Text Transfer Protocol (HTTP)-based botnet detection are discussed (see section IV-B for an explanation of this protocol). Eslahi et al. [23] do not use any botnet for their claims, but instead summarize previous surveys and taxonomies.

The presently most complete overview of the botnet phenomenon has been published by Khattak et al. [9]. The taxonomy defined therein introduces categories for every aspect of botnets and presents a comprehensive list of members for each category, which is supported by selected botnet examples. Although a detailed list of botnet tasks and topologies, and a thorough list of C&C protocols is presented, the relation between bot behavior and network communication is not drawn.

Khan et al. [10] have concentrated on spamming botnets. In addition to an overview of botnet topologies, this survey

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

TABLE I
SUMMARY OF RELATED PUBLICATIONS AND THEIR COVERAGE.

Reference	Type	Topologies	C&C Protocols	Botnet Tasks	Network patterns	Number of Botnets[1]
[22]	Survey	∼	–	∼	–	0 (4)
[23]	Survey	✓	∼	∼	–	0 (0)
[9]	Taxonomy	✓	∼	✓	–	0 (32)
[10]	Survey	∼	∼	∼[a]	–	15 (7)
[11]	Survey	∼	∼	∼[b]	–	33 (1)
[21]	Detector	–	∼	–	∼	1 (6)
[19]	Detector	∼	–	∼	∼	2 (1)
[20]	Detector	–	–	∼	∼	0 (19)

✓ Topic covered in detail. ∼ Topic covered in part or discussed briefly. – Topic not covered.
[1] Number of analyzed botnets and number of mentioned botnets, e.g., used as an example, in brackets.
[a] Focus of [10] is spam distributing botnets.
[b] Focus of [11] is DDoS.

has a detailed description of multiple spam related botnets. Examples of covered botnets are *Rustock*, *Storm*, and *Waledac*. The main or primary objective of those botnets is to distribute spam, which is the massive distribution of unsolicited emails. The purpose of those messages is either advertising or spreading malware. A more detailed analysis of these botnets is included in section V.

Furthermore, Khan et al. [10] provide an overview and analysis of different spam and botnet detectors. While proposed techniques include spam detection in botnet detectors, a more general network view is not presented.

A similar work by Hoque et al. [11] covers DDoS related botnets, with a focus on topologies. Like [10], this survey also covers botnets have been observed on the Internet in detail. Covered botnets include *Sinit*, *Phatbot*, *Rustock*, *Storm*, and *Waledac* (see also section V). Those botnets contain the ability to execute DDoS attacks during which at least as many requests as needed are targeted at a web-service for the sole purpose of disrupting the operation of said web-service. In addition to those botnets, Hoque et al. [11] cover a number of non DDoS related mobile botnets for comparing stationary and mobile botnets. This work concludes with DDoS and botnet detection techniques. Some of those techniques try to infer botnets from detected DDoS attacks.

Gu et al. [21] have proposed a botnet detector which is intended to detect botnets without the need for training or signatures. It is based on a basic infection dialog that consists of inbound scan, inbound infection, egg (executable binary) download, and outbound scan for C&C communication. This detector uses and enhances an existing Intrusion Detection System (IDS) to match network events to rules generated from the infection dialog. This approach is limited by the need for unencrypted network communication, very basic communication patterns, the need for multiple bot infections in the local network, and the requirement for active botnet propagation.

Gu et al. [19] have improved the previously mentioned technique to become independent of the C&C model, the botnet topology, and the infection model. This detector splits the measured network traffic into activity traffic, which is network traffic needed to accomplish given botnet tasks (e.g., scanning, spamming, exploits), and control traffic, which is C&C communication. Detection is achieved by cross correlating both kinds of traffic, i.e., finding multiple hosts that perform the same attacks while communicating with a common botmaster. While this work has improved on the previous detector by tolerating encrypted network traffic, multiple bot infections are still needed and local bots attacking local targets are not considered.

Ashfaq et al. [20] have extended the previously discovered infection dialog with passive botnet infections by considering spam in addition to network scans. One drawback of previous detectors is the need for observing complete bot life cycles. This deficiency is overcome by this detector with a Bayesian network. A Bayesian network represents a set of random variables and their conditional probabilities via a directed graph. In [20], the proposed botnet life cycle is represented via such a graph. The conditional probabilities are learned from training data, and the trained model then used to infer if a host is a bot.

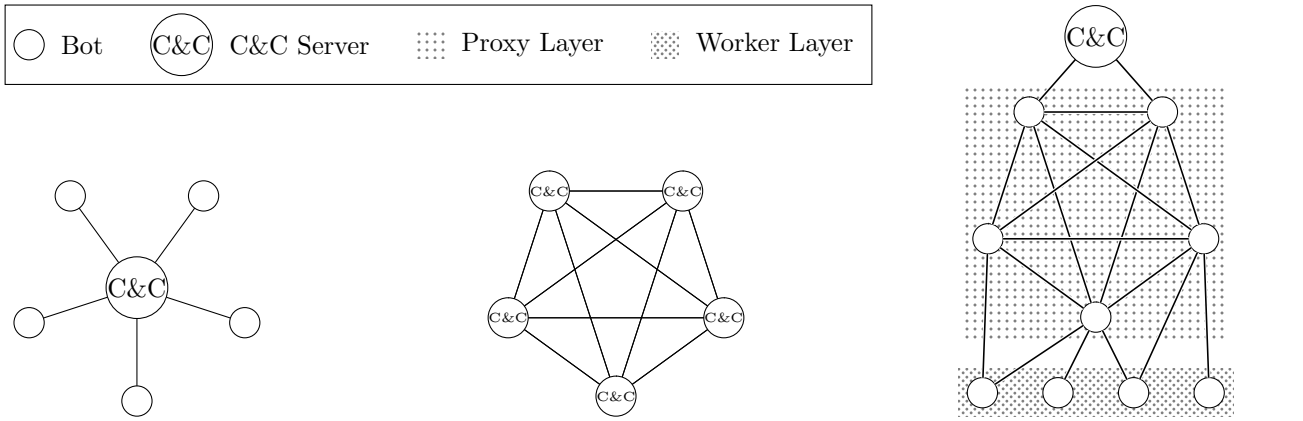
Since Ashfaq et al. [20] conclude that the life cycle will stay the same in the future, but events will evolve, the part responsible for generating the events is interchangeable.

A qualitative overview of these publications can be seen in table I. This table categorizes these papers into *survey*, *taxonomy*, or *detector*. Additionally, the table contains a rough assessment regarding the covered topics *topologies*, *C&C protocols*, *botnet tasks*, and *network patterns*. Furthermore, the number of analyzed or mentioned botnets can be seen, which is an indicator if conclusions have been drawn from existing botnets or previous surveys.

Common to all the above mentioned works is that the communication behavior of botnets is only observed from a topology and protocol perspective. This work adds a more in depth look into botnet C&C communication by providing the following contributions:

1) An in depth look into existing botnet topologies from a communication perspective. The different topologies and variations thereof are explained in detail including advantages, disadvantages, and challenges. Additionally,

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/



(a) Centralized botnet: One or multiple C&C server(s), multiple bots.

(b) Fully meshed P2P botnet: No dedicated C&C server, every bot can send commands.

(c) Hybrid P2P/C&C botnet: Combines P2P and centralized.

Fig. 2. Botnet topologies

references to existing botnets are included for every variation.

2) Detailed descriptions of C&C and operational protocols. Protocol descriptions include possible topologies, benefits, drawbacks, references to existing botnets, and historic context.
3) A categorization of existing old and new botnets from a network communication perspective. The botnets were specifically chosen to provide a high diversity of different botnet purposes, topologies, protocols, and complexity.
4) A novel taxonomy of generalized botnet communication patterns. These patterns are sequences of messages which have to be exchanged in order to perform the intended botnet tasks. This taxonomy consists of scenarios covering possible botnet tasks.
   Every scenario is explained according to the corresponding tasks, with detailed description of the needed communication patterns and examples of botnets fitting those. The pattern descriptions are independent of topologies or protocols in use.
5) A framework for building network-based botnet detectors. The taxonomy of botnet communication patterns contains UML sequence diagrams visualizing the message sequences in a standardized way ensuring interoperability and future extensibility. These sequence diagrams are independent of used communication protocol, encryption, and topology, enabling building future network-based botnet detectors.

III. Botnet Topology

As mentioned in the introduction, a botnet consists of several bots, which are infected hosts used to execute instructions given by a botmaster (see also fig. 1). To accomplish these tasks, a C&C server and bots are needed. Depending on the specific botnet, bot functionality and a C&C server may sometimes reside on the same physical host. Likewise, the functionality of a C&C server may be distributed across several physical hosts.

From a topology perspective the communication between a C&C server and bots can be organized as centralized, decentralized, and a combination of decentralized and centralized, called hybrid topologies, as categorized by [9]. In the remainder of this work these topologies will be denoted as *centralized*, *P2P*, and *hybrid*, respectively.

An overview of these topologies can be seen in fig. 2.

A. Centralized

The simplest botnet communication layout is the centralized topology. This topology uses a central dedicated C&C server with every bot connecting directly to this server (see fig. 2a). According to [24] this topology is easy to set up, has low latency, and high scalability. It is easy to set up since there are no special requirements with respect to the protocol. Therefore, simple protocols such as Internet Relay Chat (IRC) (see section IV-A) or HTTP (see section IV-B) can be used. The low latency and high scalability is caused by the simple network structure where commands are relayed directly from the C&C server to every bot.

The advantages of the centralized topology come at the cost of low robustness, which is caused by the C&C server being a single point of failure. Therefore, in order to take down the whole botnet, only the C&C server has to be taken down. This can be mitigated up to a certain level by using replicated C&C servers instead of one C&C server.

The remaining problem is that addresses of C&C servers need to be hard-coded into the botnet. Encrypting or obfuscating the program code is only of temporary value since decrypting or de-obfuscating takes a limited amount of time. Additionally, the C&C server can be detected by observing the network traffic of a bot. This can be mitigated by using a Domain Generation Algorithm (DGA).

DGAs generate different domain names derived from a changing input [25]. Most DGAs use the current time as the algorithm's input, therefore every bot in the botnet must have a synchronized system time. Depending on the used algorithm it suffices to be accurate to the nearest hour or day. If this were not the case, every bot would generate a different domain name and thus reliable communication with the C&C server would be impossible. Botnets solve this problem by either configuring

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

the computers the bot executable runs on to synchronize time or by querying popular web sites [25].

A further drawback of this system is that after figuring out the algorithm, everyone can take over the botnet by reserving a domain name the DGA will generate at a specific time in the future [25]. The domain registration problem can be mitigated to a degree by using a non-deterministic input into the DGA. Botnets have been observed in the past which used trending topics on twitter or currency exchange rates. Instead of registering domain names in advance, whoever is fastest in registering the new domain name can control the botnet [26].

Besides hiding the C&C server, this technique can also be used for links in generated spam emails. With this technique Uniform Resource Locator (URL)-based spam detection can be defeated [25]. A URL (web address) is a reference to a web resource [27].

URL-based spam detection works by constructing a list of URLs from known spam messages. URLs in emails can then be compared against this list and if a positive match is found the email be marked as spam. If a DGA is used for generating these URLs, the list creation can not keep up with the newly generated URLs.

Another way to mitigate the low robustness of a centralized topology is to create a *fast-flux* network. In a fast-flux network the Domain Name System (DNS) is utilized as a multiplexer and load balancer. DNS is a hierarchical system and accompanying protocol to associate information with domain names. This system is commonly used to associate domain names with Internet Protocol (IP) addresses (host record) [28]. Therefore, DNS provides a way to use words that can be remembered more easily instead of numbers. Additional uses include associating multiple domain names with the same IP address providing the possibility to use the same host for multiple different web pages.

It is also possible to assign multiple IP addresses to the same domain name enabling the use of multiple hosts for a single web page. This mechanism is used in a fast-flux network, where multiple bots register for one DNS host record. Upon resolving the record, one or several of these bots are returned. Instead of connecting directly to the C&C server these bots are used as intermediate hosts which in turn relay the data to the C&C server. Therefore, only these so called proxy bots know the real C&C server. Since registering and de-registering servers can be done at any time, this method can be used to cycle through multiple servers quickly [9]. This increases the effort needed to shut down the botnet. The fast-flux method shifts the single point of failure to the DNS server. This can also be remedied by additionally using the botnet as authoritative name server with the same cycling scheme, which is called *double-flux*.

Later variants of the *Zeus* botnet use a centralized architecture employing a DGA [29]. Another example is *Conficker*, which generates 50000 domains per day in version C. This high number of domains helps in distributing the load on the C&C servers and makes pre-registering the domains a logistically challenging task [30]. Examples for double-flux networks are *Storm* [31] and *Waledac* [32]. A more detailed explanation of these botnets can be seen in section V.

B. P2P

To further improve the resilience against take down or network failure a P2P topology can be used. This topology consists only of bots where every bot can be a potential C&C server. Every bot is connected to at least one other bot and commands can only reach the whole botnet if every bot has the ability to relay the commands to directly connected bots (see fig. 2b).

In a fully meshed botnet every bot is connected to every other bot. This ensures a low communication latency since messages do not need to be relayed through additional bots in order to broadcast a message to every bot. Furthermore, fully meshed botnets feature a high robustness, because removing an arbitrary number of bots from the botnet does not disrupt the communication. Since a very high number of network connections is needed for larger botnets, this approach is not scalable. Depending on the type of protocol, a fully meshed botnet may have a maximum number of bots due to operating systems limiting the maximum number of Transport Control Protocol (TCP) sockets. TCP is a transport protocol capable of reliably transferring streams of data between two hosts [33].

Additionally, the number of needed connections increases the visibility of the botnet. Furthermore, adding and removing bots due to changes in the network require a high number of coordination messages. Therefore, most P2P botnets are not fully meshed [34].

As mentioned by [24], P2P topologies are hard to implement. This is caused by the challenges of finding the initial peers and reliably distributing commands to every bot.

Finding peers can be solved by inserting a hard coded list of known peers into the bot executable. Another way to store this initial list is to utilize cache servers of existing P2P application networks. Since the bot executable has to be distributed on the Internet in order to form the botnet and the cache servers are publicly accessible, researchers can extract this initial peer list. Therefore, the single point of failure, as has been discussed above for the centralized topology, is shifted to this initial peer list. In order to get rid of publicly available peer lists, a different solution for finding the initial peers is to randomly scan the Internet for peers.

The protocol has to take care of reliably distributing commands through a botnet with a P2P topology. As mentioned earlier, botnets do not use fully meshed topologies. This is why bots must support message relaying in order to broadcast messages to every bot. Existing P2P protocols already provide all the functionality needed. If a neoteric protocol or an existing non P2P protocol is used, then reliability and routing functionality has to be added. This will be discussed in detail in section IV.

Not every computer is reachable from the Internet. This can be caused by firewalls or Network Address Translation (NAT) which hide several computers behind a single router. Therefore, some botnets classify bots into the categories super-nodes and NAT-nodes. Upon infection the connectivity is detected and later only the NAT-nodes are used for malicious activity, while the super-nodes are only used as botnet C&C communication relays. This way of deploying a P2P topology was also explored in [35].

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

Examples for botnets using a P2P topology with hard coded peer lists are the *Zeus* botnet, which falls back to centralized if no peer is reachable [36], and *Sality* (version 3 and 4), which has also a fallback to a centralized topology [37]. *Phatbot* solved the peer list problem by utilizing cache servers of a file sharing platform for the initial peer list [38]. Examples for random scanning botnets are *Sinit* [39] and *Conficker* [40]. *Zeroaccess* divided the network into super-nodes and NAT-nodes [41]. For a more detailed explanation of these botnets see section V.

C. Hybrid

As can be seen above, P2P topologies exhibit weaknesses (e.g., possibility of total disruption by removing parts of the network, non-reliable transmission of commands, finding peers, full take over via a single peer) that need to be overcome via careful design. Another way to remedy those weaknesses is to combine P2P and centralized topologies to get the benefits of both worlds [35]. Instead of the bots connecting directly to C&C servers, an additional proxy layer consisting of bots connected in a P2P topology is added. To lower the visibility of this proxy layer a third layer consisting of bots which execute the botnet tasks is added. A schematic overview of this implementation can be seen in fig. 2c.

Determining if a bot becomes a worker bot or a proxy bot can be done, as mentioned earlier for the P2P topology, based on the Internet connectivity properties of the bot. At the cost of message latency, additional layers can be added to further increase the protection of the C&C server against detection.

Another way to implement a hybrid botnet is to use a centralized topology for one part of the C&C communication and P2P for another part. For example, P2P could be used to bypass DNS for an otherwise centralized topology.

Legacy can be another reason for using part P2P topology and part centralized topology. One way to continue using existing web servers is to use P2P just for exchanging commands and a centralized topology for data transfer.

Examples for botnets using a hybrid topology are the *Miner* botnet [42] and later versions of the *Zeus* botnet [43]. *Storm* replaced DNS by using an existing P2P network for finding C&C servers [44]. Version 3 of the *Sality* botnet uses a hybrid network where the P2P part is responsible for exchanging commands, while centralized web servers are used for downloading the data. Since this centralized component represented a single point of failure, version 4 migrated to a P2P only network [37]. A more detailed explanation of botnets can be seen in section V.

IV. Botnet Protocols

Central to a botnet is its communication, which is needed to remotely control the bots. As has been discussed in section III, the layout of a botnet's topology can be mapped to one of three alternatives: centralized, P2P, and hybrid. The different topologies pose different restrictions on the communication protocols that can be used. Furthermore, protocols can be tailored to special needs such as stealthiness, which can be achieved by creating a protocol that mimics legitimate traffic or by reusing an existing protocol. Another important factor is the ease of implementation. Botnets often reuse existing implementations, therefore simplifying development. A different driving factor can be network restrictions. For example, the usage of the IRC protocol for botnet C&C communication is declining since most corporate networks do not allow IRC.

According to [9] two main categories of botnet C&C protocols exist: i) existing protocol, which means (re)using an existing application protocol that was designed for a different purpose or ii) a neoteric protocol implemented specifically for the respective botnet. In addition to the C&C protocol, a botnet needs to use additional protocols, to fulfill the desired objectives and tasks.

A. IRC

The IRC protocol was initially developed for Internet chat applications. It is a text-based protocol that allows clients to communicate with a server which is responsible for relaying the chat messages to other clients and servers. Messages can be exchanged either between a client and a group of clients residing in a so called channel or between two clients. Channels can be password protected, allowing botmasters to prevent others from taking over the botnet [45]. Additionally, file transfers are supported, allowing for the distribution of additional binaries, configuration files, or updates [46]. The IRC protocol uses a centralized topology, is easy to implement, has low latency, and is widely used on the Internet.

IRC has the disadvantage that it can be blocked easily by security devices like firewalls. Furthermore, it is not common in business networks. Later botnets with centralized topology use different protocols (e.g., HTTP).

One of the earliest botnets called *PrettyPark*, which appeared in 1999, uses a crude implementation of the existing IRC protocol for C&C and is based on the idea of IRC bots [47]. Later botnets, such as *GTBot*, utilize the scripting utilities of an IRC client called mIRC [47], thereby limiting the needed development time. Even later botnets, including *Agobot*, the predecessor to *Phatbot* [38] (see section V), still rely on IRC.

B. HTTP

HTTP was designed for delivering web pages from web servers to web browsers. It is a request-response protocol, meaning that a client can make a request, which is answered with a response from the server [48].

As main consequence, group communication is not possible with this protocol. Additionally, HTTP has a higher latency than IRC, because every bot has to specifically request commands from the server. Like IRC, HTTP is also easy to implement since several server and client implementations are available for free or as open source. Examples include nginx[1] and Apache HTTP Server[2], which are both open source web server implementations.

If every bot in the botnet implements a HTTP server and client, HTTP can also be used in a P2P topology. Since HTTP

[1]https://nginx.org/
[2]https://httpd.apache.org/

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

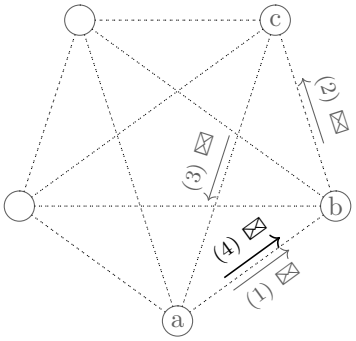


Fig. 3. P2P botnet: Duplicate message. Message delivery: (1) from node *a* to node *b*, (2) forwarded to *c*, (3) back to *a*, (4) again to *b*.

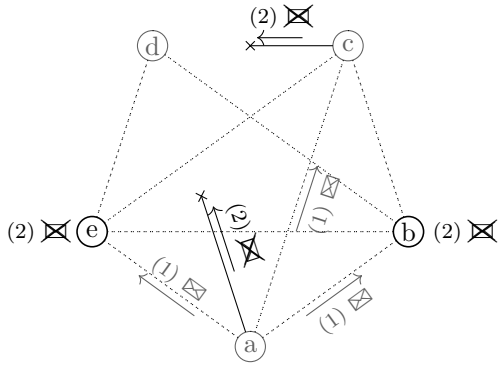


Fig. 4. P2P botnet: Lost message. Message delivery: (1) node *a* sends message to every other node; node *a* is not connected with *d*, (2) node *c* and *d* are not connected; *e* and *b* miss forwarding the message to *d*.

was designed for a centralized topology, additional care has to be taken to prevent loops. Such application layer loops could flood bots with replicated messages. This can be caused for instance by every bot forwarding a message to other bots, which in turn could forward the message back again (see fig. 3). Additional topics that need to be addressed are reliable communication across the botnet and finding other bots. An example of unreliable communication can be seen in fig. 4. In this example the botnet is not fully meshed and additionally bots miss forwarding a message. These P2P specific needs have to be addressed in an additional network layer on top of HTTP.

An example solution can be seen in fig. 5. In this model additional layers are inserted between the *application layer* (HTTP) and the *payload* (the actual botnet command). The

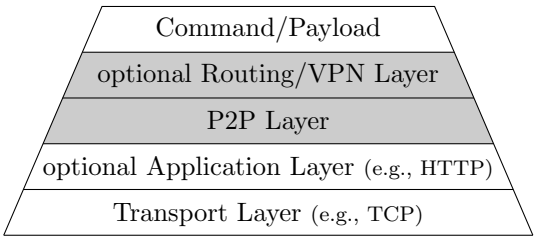


Fig. 5. P2P botnet protocol stack with non-P2P capable application protocol and optional routing/VPN layer. In case of a neoteric protocol, the application layer is not used.

P2P layer can be used to add information needed for reliably sending messages to every node and preventing the above mentioned message flooding problem. Optionally, an additional *routing layer* can be used for additional information which allows sending a command to a specific bot without knowing the needed network path in the P2P network. Additionally, messages can be routed through an arbitrary number of bots, raising the difficulty level of detecting the message origin. The reliability problem could also be solved with this *routing layer*.

The main benefit of using HTTP is that it is predominantly used on the Internet, and therefore communication via HTTP blends into normal traffic patterns.

Examples for P2P-based botnets that use HTTP for communication are *Blackenergy* [49], *Miner* [42], *Regin* [50], early versions of *Sality* [37], *Sinit* [39], and *Waledac* [32] (see also section V).

C. SMB

Another protocol that conceals communication within typical traffic patterns in home and business networks is the Server Message Block (SMB) protocol. Like HTTP, SMB is a request-response protocol and works by exchanging messages between a client and a server. First a client has to authenticate itself to the server with user credentials or as an anonymous user. After authentication, a client can request a list of available shares, which are pools of files or services, or access files or services (e.g., shared printers) [51].

The SMB protocol is mainly used to provide access to files and printers over a network. Additionally, the SMB protocol implements named pipes, which are a mechanism for Inter-Process Communication (IPC), even across the network. IPC can be used by different processes to exchange messages or call specific functions.

Another way to use the SMB protocol is for infecting other hosts. If a computer shares files via the SMB protocol, and those files are writable, then a bot can inject malicious code into these files. To infect a computer with those files, a user has to open them. Therefore, this is called passive infection (see also section VI-C).

The SMB protocol is often blocked at the gateway that provides access to the Internet. Hence, this mechanism is used mostly for communication in local networks.

Examples of botnets using the SMB protocol are *Regin* [50] and *Duqu 2.0*, which is also capable of emulating a SMB server for collecting passwords from authentication attempts [52]. Additionally, *Phatbot* can propagate by infecting files shared via the SMB protocol [38], [53]. More details about the botnet examples can be seen in section V.

D. P2P Protocols

Existing P2P protocols which were originally developed for file sharing and collaboration purposes can be used for botnets with a P2P topology. These can be used to form a separate P2P network or to hide as part of one of the existing networks (see fig. 6).

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

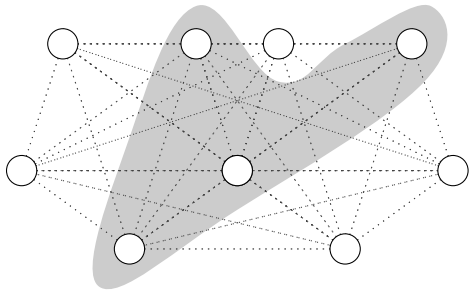


Fig. 6. P2P botnet (gray) utilizing parts of an existing P2P network for C&C.

An example for an existing protocol is WASTE[3], a protocol originally designed for collaboration [55]. Another popular class of P2P protocols are based on Kademlia. These protocols can be used to store values or hashes in a distributed and fault tolerant way [56].

A botnet using parts of an existing network is *Phatbot*, which uses Gnutella cache servers for finding its peers. The C&C part of *Phatbot* uses the WASTE protocol, but without encryption, therefore simplifying the communication [38]. Examples for Kademlia based protocols are the P2P variants of *Zeus*, that use the same distance metric as Kademlia [43], and *Storm*, which uses the Overnet protocol instead of DNS [44]. Overnet is a Kademlia based P2P network and protocol, that includes its own routing layer and addressing. It is mainly used for file sharing [57].

E. Neoteric Protocols

As has been mentioned in the beginning of this section, there are also botnets that use neoteric protocols for C&C. Most of these protocols are User Datagram Protocol (UDP) or TCP-based. UDP is a simple transport protocol that offers datagram services, i.e., unreliably transmitting single packets [58]. TCP is a transport protocol capable of reliably transferring streams of data between two hosts [33]. Additionally, Internet Control Message Protocol (ICMP)-based C&C channels have been observed. ICMP is a protocol for exchanging control messages (e.g., destination host unreachable) [59].

Depending on the used botnet topology and base protocol, care has to be taken during the protocol design. For P2P topologies, the protocol has to be reliable and prevent flooding the network with duplicate messages (see section IV-B). Additionally, if UDP is used, the reliable transmission of messages between two hosts needs to be implemented. This can be achieved by prepending headers to the actual command contained in a single UDP packet (see fig. 5). Duplicate commands can be detected using a command id from the command header. Reliability can be implemented using retransmissions and the optional sending of acknowledgments. This solution is faster, more lightweight, and has less protocol overhead compared to TCP. A higher number of concurrent connections can be operated compared to TCP as no dedicated connections need to be maintained. Moreover, this solution can help to work around limits that are imposed by the operating system with respect to the maximum number of concurrent TCP connections.

The payload can be either binary or text-based. Since no text parsing is needed, binary protocols are easier to implement. Additionally, the text-based protocols can be easier reverse engineered since the commands have to be listed in the bot binary and thus can be extracted with low effort.

The following paragraphs contain examples of neoteric protocols which were invented for different botnets. For a more detailed description of the listed botnets see also section V.

One of the first botnets using a neoteric protocol is *trinoo*. C&C communication is split into two parts, where one part uses a password protected, American Standard Code for Information Interchange (ASCII)-based, self-developed protocol via TCP, while the other part uses UDP packets containing short ASCII strings [60]–[62].

An example for a botnet that uses a binary protocol is the *Slapper* botnet. *Slapper* uses UDP for the transport and writes C structures directly into the payload [63]. A C structure is a container for variables which are arranged directly one after another in memory. *Slapper* uses the above mentioned technique of adding headers to the packet payloads to handle reliability and routing (transport layer, P2P layer, routing layer, command in fig. 5) [63].

The *Conficker* botnet is only capable of distributing and executing binaries. This allows for a much simpler neoteric P2P-based protocol. Bots can only query if a newer binary is available and download it if this is the case [40]. With these restrictions in mind the duplicate message problem is solved with the binary version since the newest version always wins and binaries are only exchanged if there is a newer one. Since every bot periodically tries to find new versions, the reliability problem is also solved.

Another example for a botnet with a simple P2P protocol is *Sality*. Like *Conficker*, *Sality*'s main purpose is to distribute malware binaries. In addition to the file sharing capabilities implemented in *Conficker*, the *Sality* protocol implements a peer exchange and a command for marking a bot as super peer. A super peer is publicly accessible from the Internet and can therefore be contacted by other bots. Peer lists consist only of super peers. Similar to *Conficker*, the duplicate message problem is solved with a version of the files to be downloaded and the reliability problem with every bot trying to download new versions periodically [37].

F. Communication Hiding & Obfuscation

1) Covert Channels: One drawback of neoteric protocols (see section IV-E) is that they are not commonly used in networks. Therefore, they can be easily detected. Existing protocols have the drawback that botnets cause communication that does not exist during normal operation. Both problems can be remedied with the use of covert channels. Covert channels are used to hide the existence of communication [64]. This is achieved by using means of communication that are not intended for communication use. For instance unused header bits in, e.g., TCP or IP, can be used to transmit information [64].

[3]WASTE is a reference to the novel "The Crying of Lot 49" [54]

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

The *Tribe Flood Network* uses the identifier field of ICMP echo reply packets as a command for C&C [60], [62]. Despite the effort to hide the communication, these packets can be easily detected and manipulated by looking for ICMP echo replies with missing ICMP echo requests.

2) Encryption: Encryption allows to hide the transmitted commands and even the used protocol from detectors. This defeats every type of network-based detector that needs to inspect the payload. Common to every encryption algorithm is that with the help of a single key or multiple keys clear text can be transformed into cipher text and vice versa. Encryption algorithms can either be stream based or block based. With stream based encryption algorithms it is possible to encrypt single bytes, while block based encryption algorithms can only encrypt blocks of fixed size.

Furthermore, algorithms can be either symmetric, meaning the same key is used for encryption and decryption, or asymmetric, where two complementing keys are used [65].

The easiest way to encrypt data is with the exclusive or (XOR) operation, where the exclusive disjunction between every bit of the clear text and every bit of the key is calculated. Repeating this operation on the resulting cipher text results in the original clear text. This is a simple symmetric stream-based encryption algorithm. One of the drawbacks is, that using the same key with the same clear text results in the same cipher text. This means, that if a botnet uses a predefined key and fixed commands, the resulting encrypted commands can again be identified. Even if the same key is used to encrypt multiple samples, it is possible to calculate the original key from just the cipher texts [65].

These problems can be solved by using an endless key, which is the idea behind stream ciphers which create an endless pseudo random sequence which is determined by the key. This sequence is then used with the XOR operation on the clear text. An example for such a cipher is Rivest Cipher 4 (RC4) [66].

Block ciphers work in several rounds on the clear text, which consist of different combinations of XOR operations, substitutions, and permutations. Examples for symmetric block ciphers are Advanced Encryption Standard (AES) [67], Camellia [68], Rivest Cipher 5 (RC5) [69], Tiny Encryption Algorithm (TEA) [70], and Extended Tiny Encryption Algorithm (XTEA) [71].

Asymmetric ciphers are based on numeric problems that are hard to solve (e.g., prime factorization). One example is Rivest-Shamir-Adleman (RSA) [65]. Since these algorithms are computationally expensive, they are used mainly to exchange keys for a symmetric encrypted connection and for signing.

Without the knowledge of the key it should not be possible to derive the clear text from the cipher text. Exceptions can arise if errors in the algorithm or errors in the implementations are found or requirements of the algorithm are violated. Therefore, it is advisable to use existing implementations like Transport Layer Security (TLS). TLS handles encryption and optionally authentication. One commonly used protocol is Hyper Text Transfer Protocol Secure (HTTPS), which is HTTP layered on top of TLS [72], [73]. Several libraries exist that implement all the needed functionality.

Examples for botnets using flawed implementations of encryption algorithms are the *Tribe Flood Network 2000*, which caused every network packet generated by the botnet to end in ASCII 'A' characters [60], and *Blackenergy 2*, which reused the same key for every message [74]. *Blackenergy* version 3 changed this implementation to using HTTPS instead [75]. Another example which uses HTTPS is *Adwind* [2].

Sality version 3 and 4 use RC4 encryption for P2P protocol messages. Since the key for encryption is always transmitted in the same message, this is used solely to obfuscate the communication and defeat signature-based detection approaches [37].

3) Multiple Protocols: To further increase stealthiness of the botnet instead of a single C&C protocol, different protocols within the botnet can be used. This can be achieved by building a variable communication layer capable of using different communication protocols.

Duqu 2.0 is a botnet that is capable of using HTTP, SMB named pipes, HTTPS, or a neoteric TCP protocol for C&C traffic. Those protocols can be combined with several compression and encryption methods [52]. With this vast choice of protocols the botnet can use the most predominant protocol of the target network.

4) Compression: Another way to obfuscate communication is to use compression. A compression algorithm reduces the input according to an algorithm (e.g., replace repeating sequences with sequence and number of repetitions, use index into a dictionary instead of the sequence itself). Unlike encryption, no key is used and thus everyone knowing the compression algorithm can decompress the data. Therefore, compression can only obfuscate the data.

Used algorithms include zlib, Lempel Ziv Jeff Bonwick (LZJB), Lempel Ziv Free (LZF), Fast Lempel Ziv (FastLZ), Lempel Ziv Oberhumer (LZO), and bzip2 [76].

Examples for botnets using compression are *Zeus*, which uses zlib [43], and *Duqu 2.0*, which can use multiple different algorithms [52].

5) Steganography: An additional obfuscation technique is steganography, which means hiding information in containers that are not meant for communicating. This technique is similar to covert channels, but instead of network protocols files are used as carriers (e.g., images, videos, documents) [64].

The botnet *Duqu 2.0* uses this technique if unencrypted HTTP is used as protocol for C&C communication [52]. Nagaraja et al. [77] have proposed a botnet that uses an existing social network as C&C channel and hides the communication in Joint Photographic Experts Group (JPEG) pictures. Since this method relies on the user uploading images, the number of possible transmitted messages is limited.

A more novel approach by Compagno et al. [78] hides the C&C communication in text messages exchanged via social networks by users. This approach encodes the C&C messages with control codes that are not displayed by web browsers. Additionally, information is encoded by rearranging code combinations which result in the same visual representation. Both approaches piggyback the C&C communication onto

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

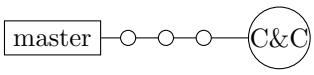


Fig. 7. Stepping stones between botmaster and C&C server, which improve stealthiness of botmaster.

legitimate user traffic defeating most network based botnet detection mechanisms.

6) Stepping Stones: Since botnets are mostly used for malicious activities, a botmaster needs to stay as hidden as possible. One way to increase the secrecy is to use stepping stones, which are hosts that are used as intermediate hops for communication between the botmaster and the C&C server (see fig. 7).

In a P2P topology this can also be achieved with bots in the network. In order to simulate the behavior of having intermediate hosts between the botmaster and the source of the control commands, an additional routing layer can be used (see section IV-B). This routing layer adds the capability to specify a number of intermediate hosts that the command needs to be relayed through until it is allowed to reach its target destination.

The *Slapper* botnet uses this mechanism by choosing a random next hop at each step [63].

G. Operational Protocols

In addition to C&C traffic, botnets also need to communicate with other services in order to achieve their objectives. This is achieved via operational protocols, which can be distinct from the used C&C protocol and topology.

In order to send spam, which is unsolicited email traffic, botnets need to be able to communicate via Simple Mail Transfer Protocol (SMTP). The SMTP protocol is a very simple request-response protocol, where anyone can connect to a mail server and submit emails [79]. Most servers require authentication only for sending emails which are intended for a different server, which is the main cause for spam. Therefore, many email providers try to block emails that originate from home IP addresses.

HTTP and HTTPS are also needed for implementing click fraud. Click fraud is a scheme, that uses bots to access advertising URLs, which in turn generates revenue [9].

In addition to click fraud the HTTP protocol, as well as UDP or TCP is also needed to execute DDoS attacks. A DDoS attack is an attack, that attempts to overload a networked service. This can be achieved by sending large numbers of packets to the target host or network. Since bandwidth is limited, this will result in a service outage by consuming all the bandwidth with attack traffic. Another way to overload the service, is to send computationally intensive requests to the targeted application. These requests will use up all the available processing power rendering the service unresponsive [11].

V. Botnets

Botnets are used for varying purposes including Remote Administration Tool (RAT), DDoS attacks, distributing spam, click fraud, service disruption, distributing malware, or as a general purpose Content Delivery Network (CDN). Those objectives, the distribution, and the maintenance of the botnet have different communication needs. To fully understand the communication patterns of botnets, a highly diverse list of botnets is analyzed in this paper.

The previous botnet surveys [10], [11] are limited to botnets with a specific targeted purpose, while more general surveys [9], [22] only reference or mention botnets as examples. In contrast, the focus of this work is on the specification of general models for botnet communication. As a consequence, this taxonomy is built on top of an intentionally diverse list of botnets in order to extract the communication commonalities of a variety of botnets.

To compile a taxonomy of task dependent communication patterns, the botnets have been analyzed according to:

- Topology of the botnet for determining if the topology has an influence on generalized communication patterns.
- Purpose of the botnet.
- C&C protocols used and the relation to the purpose of the botnet.
- C&C communication related obfuscation and hiding techniques and the influence on the communication patterns.
- Tasks that can be executed.
- C&C communication needed for those tasks.
- Network communication needed for the botnet tasks.

The following list of assorted botnets (see also table II) includes botnets with varying degrees of sophistication, different communication approaches, various topologies, and differing goals. Botnets in the list were selected as representatives of their class, either because they were the first including some new methods or because they use an especially sophisticated variant of a particular technique. Further botnets have been selected that went through a radical evolution during their lifetime. This evolution represents a general botnet trend. Furthermore, old and new botnets have been chosen to inspect if and how communication patterns evolved over time.

1) *Adwind* [2], [80]: A special feature of this botnet is that it is marketed as software as a service. This means, that the software is hosted by a provider and one can only buy a subscription that lasts for a limited time. Therefore, *Adwind* requires the use of a centralized structure with a limited number of C&C servers for license checking. So despite the fact that most modern botnets use P2P or hybrid topologies this botnet still uses a centralized structure with a limited number of C&C servers. This licensing scheme could also be achieved with a hybrid topology, but this would increase the implementation and operation costs. The use of a hybrid topology also imposes restrictions on the minimum size of the botnet since a number of bots are needed to relay communication. This makes the business model of software as a service for a botnet too expensive to use it in a targeted attack.
   Since there are also rebranded, copied, and cracked versions, many more C&C servers than the initial ones exist. This is one reason why blacklists for blocking C&C servers are only of limited value. *Adwind* is mainly

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

TABLE II
OVERVIEW OF COVERED BOTNETS, FEATURES AND REFERENCES

Botnet	Year[1]	C&C Protocol	Topology	Encryption	Comp.	Purpose	Related/Family	Refs
Adwind	2012	HTTPS	central	TLS	-	RAT, modular	AlienSpy, Frutas, jFrutas, Unrecom, Sockrat, JSocket, jRat	[2], [80]
Blackenergy	2007	HTTP, google+	central	v1: -; v2 RC4[a]; v3: TLS	-	DDoS, modular	multiple versions	[29], [49], [74], [75], [81], [82]
Conficker	2008	HTTP, SMB, UDP, TCP	central; later P2P, central	RC4 (P2P)	-	distribute malware	Downadup, multiple versions (A – E)	[30], [40], [83], [84]
Duqu 2.0	2014	HTTP, SMB named pipe, HTTPS, TCP	central	Camellia, AES, XTEA, RC4, XOR, TLS	LZJB, LZF, FastLZ, LZO	RAT, modular	Duqu, Flame, Gauss, Stuxnet, miniFlame	[52], [85]
Miner	2010	HTTP	central; later hybrid P2P/ central	-	-	PPI, (Bitcoin), DDoS, steal identities	multiple versions	[42]
Phatbot	2004	WASTE[b], IRC	P2P, central	-	-	steal data, DDoS, deploy malware	Agobot, Gaobot, Forbot, XtrmBot, over 1000 variants	[38], [53], [86]
Regin	2003	UDP, TCP, SMB, HTTP, HTTPS	P2P, VPN	RC5, TLS	-	modular	QWERTY	[50], [87]–[89]
Rustock	2005	HTTP	central	RC4[c], TEA	-	spam	multiple variants	[90], [91]
Sality	2003	email, HTTP, UDP, TCP	central; later P2P	-	-	load software, steal data	multiple versions	[37]
Sinit	2003	UDP	P2P	-	-	distribute malware	Calyps.a, Calypso	[39]
Slapper	2002	UDP	P2P, routing	-	-	DDoS, load software	Scalper, Cinik, Unlock, multiple versions	[63], [92]
Storm	2007	Overnet[d], HTTP	hybrid P2P/ central	XOR	zlib	spam, DDoS	Peacomm, Nuwar, Tibs, Zhelatin	[31], [44], [93], [94]
Stuxnet	2005	HTTP, TCP, SMB named pipe	P2P, central	XOR	-	disrupt SCADA	Duqu, Duqu 2.0, Flame, Gauss, miniFlame	[5]
TFN[e]	1999	TCP, UDP, ICMP	central	original: -; 2000: CAST-256	-	DDoS	Stacheldraht	[60], [62], [95]
Trinoo	1999	TCP, UDP	central	-	-	DDoS	Stacheldraht, Trin00	[60]–[62]
Waledac	2008	HTTP	hybrid P2P/ central	AES, RSA	bzip2	spam, load software	-	[32], [96]
Zeroaccess	2011	UDP, TCP	P2P	RC4, custom XOR	-	click fraud, load software	Smiscer, Max++ rootkit, Sirefef	[41], [97], [98]
Zeus	2006	UDP, TCP, HTTP	central; later hybrid P2P/central	RC4	zlib	steal credentials	multiple variants, Gameover, Murofet, Licat	[29], [36], [43], [99], [100]

[1] First sample [a] Version 2 used a weak implementation of RC4 [74]. [b] Without encryption [38], [86]. [c] Key matching computer hardware of bot.
[d] Overnet is used as a replacement for DNS [44]. [e] Tribe Flood Network

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

used as a RAT in targeted attacks. A RAT allows the botmaster full control of the computer the bot executable runs on. Depending on the included plugins, *Adwind* can be used to delete files, upload files, modify files, monitor the owner of the computer including access to attached devices like webcams, exfiltrate sensitive data including login credentials, and even access all data on mobile phones. This functionality in the *Adwind* botnet was used to get access to financial institutions, monitor government officials, or simple money extortion schemes.

Besides being used as a RAT, *Adwind* also allows for other scenarios due to the modular design. Discovered plugins include a crypto currency miner, and proxies. Additionally, it contains a java obfuscater to evade detection and is able to run on OSX, Windows, Linux, and Android. Communication is protected via HTTPS and uses non-standard ports, which means that a different port than 443 is used which is suggested by the standard [73] for HTTPS.

2) *Blackenergy* [29], [49], [74], [75], [81], [82]: This botnet exists in multiple versions. Every version uses a centralized topology and a HTTP server as C&C. Early versions did not use encryption and were only capable of DDoS attacks. Later versions secure the communication with a weak RC4 implementation and add several modules enabling the botnet to be used for service disruption (e.g., routers, Supervisory Control and Data Acquisition (SCADA)).

   SCADA systems are used for controlling processes in large scale industrial control systems. Since those systems are sometimes geographically dispersed, they need to be connected to the Internet. Additionally, those systems are often connected to office networks for exporting data to accounting or updating processes remotely [101].

   Blackenergy can also be used as an Advanced Persistent Threat (APT), which is a targeted attack where a skilled adversary gains access to a network and remains there undetected. This allows for further reconnaissance and accessing different levels of the network to finally achieve the main objective (e.g., data exfiltration, interrupting normal operation) [101], [102].

   At present time the latest version uses HTTPS for communication and a module has been identified which can use the social network Google+[4] for communication.

   Blackenergy represents one of the trends in botnet evolution. It started out as a simple DDoS capable botnet with unencrypted C&C communication and utilized a centralized topology. Later on, *Blackenergy* was modularized, which allowed for additional purposes besides DDoS. During this process encryption with differing levels of sophistication was added. Despite the simple start it is currently even used in high profile attacks on the power grid [7].

3) *Conficker* [30], [40], [83], [84]: This botnet evolved very fast in 2008 and 2009, undergoing five different versions (named $A - E$). All the earlier versions received updates, upgrading bots to the latest version. Additionally, version *E* installed malware on the infected computers and removed itself on 3rd of May 2009.

   Early versions employed a centralized botnet topology, using a DGA for generating domain names. Since the DGA used the current time as a seed, bots sent HTTP requests to popular web sites for retrieving the current time. The only purpose of the central server was to distribute binaries. Since those binaries got loaded directly into the botnet, it was possible to add additional functionality.

   After the DGA was reverse engineered and the results were used to disrupt the botnet, the DGA was updated. Additionally, later versions of *Conficker* also contained a P2P component for distributing binaries. Under specific circumstances, the centralized part was switched off and only the P2P component used.

   To harden the botnet against adversaries, *Conficker* employed RSA signatures for every binary update, and used RC4 encryption for the P2P protocol. Furthermore, the P2P component used random scanning for peers. Additionally, the ports used by the P2P component were generated from the current week number and the IP address.

4) *Duqu 2.0* [52], [85]: This sophisticated botnet was first discovered by Kaspersky Lab during a security sweep within their own network. It is highly modular, can be used as a RAT, and features several techniques that make it especially suited as an APT. In order to evade detection, several encryption techniques including Camellia, AES, XTEA, RC4, and chunk-wise XOR with a fixed key can be used by this botnet.

   Additionally, multiple proxy modules are available that enable translating from any of the used protocols to HTTP, HTTPS, SMB named pipes, and a neoteric TCP protocol over different ports. Moreover, network traffic over HTTP is hidden using steganography (see section IV-F5).

   Further network traffic mangling can be achieved by utilizing the compression modules which support LZJB, LZF, FastLZ, and LZO. This APT has the unique behavior that the persistence module is only used on servers, which means that other hosts need to be reinfected after every reboot, causing additional traffic.

5) *Miner* [42]: Early versions of the *Miner* botnet use a centralized approach with several C&C servers. Later versions replace this topology with a layered structure consisting of static master C&C servers, a P2P/CDN layer, and a worker layer. Clear text HTTP over port 8080 is used for communication. Transmitted binaries are signed with an RSA encrypted Message Digest 5 (MD5) hash.

   Hashing functions are one way algorithms that transform input of arbitrary size into shorter numbers representing the original content. The hash of two identical copies results in the same number. This can be used as a shortcut for verifying file contents [65].

   The *Miner* botnet, as a Pay-Per-Install (PPI) service, is used as CDN for additional malware and for running

[4]https://plus.google.com/

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

DDoS attacks. The most used additional malware includes bitcoin miners and software capable of stealing social network identities.

The *Miner* botnet is an example of a botnet which harnesses the computing power of its victims for financial gain (e.g., mining bitcoins).

6) *Phatbot* [38], [53], [86]: This botnet is a successor of *Agobot*, which used a centralized topology over IRC. In this evolution an additional communication mechanism using a P2P topology via the WASTE protocol is added (see section IV-D for a description of WASTE). The encryption part of the protocol is removed in order to avoid the problem of key distribution, which is not a part of WASTE. The botnet utilizes Gnutella cache servers for finding its peers. Binaries needed during the infection stage and updates are transmitted over plain text File Transfer Protocol (FTP). FTP is a simple text-based request-response protocol used for transferring files between a server and a client [103].

   The botnet uses a modular architecture for its payload allowing for various malicious activities. The main goals of this botnets include stealing of data, performing DDoS attacks, and deploying additional malware.

   Phatbot is one of the earliest attempts to replace the centralized topology, which was predominant by that time, with P2P. This attempt seems to have failed since despite extensive modifications to the rest of the bot, in later version of the botnet the WASTE protocol was never updated. Additionally, *Phatbot* still contains the IRC protocol and the WASTE protocol was designed for small networks with 10-50 nodes.

   Since the source code for the original *Agobot* and *Phatbot* was published on the Internet, there are several thousand variants of this botnet.

7) *Regin* [50], [87]–[89]: Although the first known sample of *Regin* dates back to 2003, this botnet was first discovered in 2012. This is caused by the sophisticated Service Oriented Architecture (SOA), which is used to implement a highly modular botnet where every module communicates through a Virtual Private Network (VPN). With this design the functionality can be spread over several bots and changed dynamically during run time. The VPN can be transported over a neoteric UDP protocol, a neoteric TCP protocol, SMB, HTTP, and HTTPS.

   To initialize the connection between two bots, one bot sends a knocking sequence to the other bot. The two bots then negotiate which transport protocol should be chosen for C&C communication. This knocking sequence can use any of the above mentioned protocols and additionally, ICMP. With this technique, the communication is adapted to match the non malicious traffic occurring in the target network. Additionally, this technique can be used to exfiltrate data through firewalls since a protocol can be chosen that is permitted by the firewall. Communication is protected by TLS including certificates for authentication. The whole mechanism is used to build a P2P network and is mainly used as an APT.

   While *Duqu 2.0* is also capable of using different protocols between bots, *Regin* is capable of handling this automatically. Additionally, the overlaid VPN hides this complexity from the operator. *Duqu 2.0* is also the only known botnet as of today, that uses a SOA for its modules.

8) *Rustock* [90], [91]: The *Rustock* botnet is used for distributing spam. Earlier versions use HTTP for communication and encrypt the transmitted data with RC4 including a custom key exchange. Later versions replace HTTP by a TEA encrypted neoteric protocol disguised as HTTPS traffic with keys derived from the system information of the infected host.

   Every version uses a central C&C server for coordination. In case the configured C&C server can not be reached, *Rustock* tries to contact C&C server addresses generated by a DGA.

   In addition to sending spam via SMTP this botnet is also capable of using web-based mail services (e.g., windows live hotmail) with stolen credentials, which increases the difficulty of detecting the botnet by observing network traffic.

   Rustock is highly visible on the Internet since it tries to send as much spam as possible. Despite this fact, a joint effort of law enforcement, multiple security firms, and a pharma group was needed to bring the botnet down by seizing multiple C&C servers around the world at the same time.

9) *Sality* [37]: Early versions of *Sality* are used to steal information including passwords and data obtained by keylogging. The collected data is sent back to the attacker via email and no control of the infected hosts is possible. Later versions split the malware into a botnet being capable of installing additional software and a payload which provides the information stealing capabilities.

   This simple botnet uses clear text HTTP to communicate with a central C&C server. Since the URLs are hard-coded, take down of the botnet is easily achieved by shutting down the central server. To prevent this, the topology of *Sality* was changed to P2P with RC4 encrypted communication over pseudo random generated UDP ports. Bots are split into worker bots executing the botnet tasks, and super peers used solely for C&C communication purposes.

   The main purpose of the latest version of *Sality* is to load additional software for information theft and network scanning.

   Sality is designed to be take down resistant. The P2P code keeps a list of peers, which is refreshed every 40 minutes. This makes it impossible to inject bad peers. For security measures, these peer lists are limited to 1000 peers per bot, therefore preventing a single entity from knowing big parts of the network. Additionally, binary downloads are signed.

10) *Sinit* [39]: The main purpose of this botnet is to serve as a CDN, which is used to distribute additional malware. *Sinit* uses a neoteric protocol on top of UDP. The botnet topology is P2P and bots find peers by random scanning the Internet. UDP port 53 is used for communication, which is the same port and protocol DNS uses. This lead

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

to peer discovery being mistakenly interpreted as DNS fingerprinting.

Distributed binaries are digitally signed to prevent foreign code. In addition to the P2P protocol, the botnet includes a small web server, which only provides the bot binary for downloading, thus spreading the botnet.

Sinit is one of the early attempts of building a P2P topology with a neoteric protocol.

11) *Slapper* [63], [92]: The *Slapper* botnet is a successor to the *Scalper* botnet, which uses a P2P topology and is capable of distributing spam, executing arbitrary commands on the infected host, and performing DDoS attacks.

    The P2P protocol is made more robust and efficient compared to the predecessor. This is handled by an additional reliability layer in the protocol used for acknowledgment. An additional routing layer is capable of routing a message through several bots before it reaches the target bot. This allows for hiding the origin of messages. The routing layer can also thwart correlation based detection attempts since it enables bots in the same network to receive the same message from different bots.
12) *Storm* [31], [44], [93], [94]: This botnet utilizes a hybrid P2P/centralized architecture consisting of a central C&C server hidden behind proxy servers. These proxy servers are connected to the existing P2P network Overnet (see also section IV-D). The P2P network is used as a replacement for DNS, thus hiding from DNS-based botnet detectors. The information stored in Overnet is only 16 bytes long and encrypted with a static 16 byte key using XOR.

    Only worker bots are used for performing DDoS attacks and distributing spam, while the other bots are in charge of providing a double-flux network for the botnet in multiple layers. Files are distributed via HTTP and commands via a neoteric text-based TCP protocol. *Storm* has the capability to detect scanning attempts and launch DDoS attacks in order to prevent detailed investigation of the botnet.
13) *Stuxnet* [5]: The main purpose of this botnet is to disrupt a specific SCADA infrastructure. This botnet has only limited control abilities and uses a P2P network for updates in networks with no Internet connection, HTTP in networks with Internet connection, and can also spread and update itself via infected Universal Serial Bus (USB) drives. The combination of USB infected drives and network communication allows this botnet to even infect air gapped systems (see also section VI-C2) and offers the option to control these networks at the cost of very high unreliability and very high latencies. P2P communication is XOR encrypted with a static key and runs on top of a neoteric TCP protocol or SMB named pipes. *Stuxnet* is very sophisticated, believed to be targeted at a single industrial facility, and supposedly ordered or authored by a nation state's intelligence service.
14) *Tribe Flood Network* [60], [62], [95]: The *Tribe Flood Network* was created in late 1999 as a show case to demonstrate the possibility of a sophisticated DDoS attack tool. Early versions use ICMP echo reply packets for communication, with the command encoded in the identifier field in order to evade detection. The later version *Tribe Flood Network 2000* changes the C&C communication to a neoteric CAST-256 encrypted protocol transported via UDP and is also able to run on Windows, Linux, and Solaris. Stealthiness is further increased by omitting acknowledgments. To eliminate potential communication channel unreliability, commands are sent multiple times to the bots. The botnet can be further disguised by sending fake command packets to fake destinations. Although this botnet is one of the earliest, it already features modern hiding techniques like covert channels and encryption.
15) *Trinoo* [60]–[62]: *Trinoo* was developed in early 1999 to launch coordinated DDoS attacks. It features a C&C server component called handler and bots called agents. Commands can be sent to the C&C server over a simple text-based neoteric TCP protocol. The C&C server is password protected with a hard coded password and it tries to notify the botmaster of botnet overtaking attempts. A neoteric UDP protocol featuring simple text-based commands and custom acknowledgments is used as C&C protocol. Like *Tribe Flood Network*, *Trinoo* is also a very early attempt to create a botnet.
16) *Waledac* [32], [96]: This botnet is mainly used to distribute spam and execute additional malware on the target host. The topology used for communication is a hybrid C&C/P2P approach consisting of a master C&C server, a layer of relay nodes, and a slave node layer. Like the *Storm* botnet, this infrastructure is used as a fast-flux network. Communication uses AES and RSA encrypted HTTP requests, which are additionally compressed with bzip2. One feature distinguishing *Waledac* from other botnets is that every bot generates RSA keys and certificates. These are used to encrypt session keys, which in turn are used to AES encrypt the C&C communication. Additionally, the C&C communication is heavily obfuscated by using multiple steps like compressing, encrypting, encoding, and mangling the messages.
17) *Zeroaccess* [41], [97], [98]: The *Zeroaccess* botnet employs a P2P network for communication. This P2P network is divided into super-nodes used for file distribution and normal nodes. Super-nodes need to be reachable from the Internet, and therefore nodes are declared as normal nodes if they are behind a NAT. As a communication protocol, a neoteric UDP and TCP protocol is used, which are encrypted with RC4 or a custom XOR-based encryption. One of the main purposes of this botnet is click fraud. Additionally, this botnet is used for search engine poisoning, which is the hosting of malicious websites that contain targeted keywords. Therefore, these pages show up in search engine results, luring unsuspecting users to malicious web content.

    Zeroaccess has a very limited set of C&C commands, which can only be used for updating the bot binary or loading additional malware. The additional communication needed by the modules needs to be handled by the modules themselves. Therefore, this botnet needs multiple

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

C&C channels, each acting differently.

18) *Zeus* [29], [36], [43], [99], [100]: This botnet uses several central C&C servers in the early versions. In order to make the botnet resilient against C&C server take-downs, the botnet features a DGA, that randomly generates 1000 DNS host names per day in order to find C&C servers. Since this algorithm was discovered, the *Zeus* botnet migrated to a P2P topology. In case a bot can not access any peers via P2P, it falls back to finding a C&C server via DGA. The protocol uses UDP for messages and TCP for file transfers. Communication is encrypted with RC4 and compressed with zlib. During startup of the bot executable, peers are discovered by connecting to hosts taken from a hard coded peer list. Several thousand variants of the botnet exist, because the source code of the botnet was published on the Internet. The different variants also feature variations in the protocol, making it suitable for determining similarities in behavior although different protocols are used.

VI. Communication Patterns

A central component to botnets is the communication between botmaster and bots. As explained in sections III and IV, it is paramount for the botmaster to stay hidden. Therefore, botmasters do not communicate directly with bots but use a C&C server. Since detection of the botmaster can potentially lead to prosecution, the botmaster tries to stay hidden, using as little traffic as possible [9]. Bots typically generate additional network traffic during execution of their assigned tasks, which increases their visibility in the network. Thus, detecting botnets based on network traffic has the highest chance of success when observing traffic close to the bots. Therefore, this work concentrates on the communication patterns that can be observed at or near bots.

A. Notation

The communication patterns identified for botnets are presented in generalized communication patterns. A communication pattern is a sequence of exchanged messages needed to achieve a specific task. For example, in order to display a web page, a web browser needs to send a request to a web server. The server sends back a response containing the contents of the requested page. If, for instance, this web page contains pictures, the same request-response process has to be repeated for every picture. Web browsers use HTTP as the network protocol, which was designed to fulfill this communication pattern. A form of visual representation for such patterns is provided by UML sequence diagrams. An example of a sequence diagram representing the web browser communication pattern is visualized in fig. 8.

A communication pattern is called an interaction in UML sequence diagrams. The base components of such an interaction are lifelines and messages [104]. The participants in an interaction are represented by lifelines. In the example given by fig. 8, these are the *Web Browser* and the *Web Server*. These participants exchange messages over the lifetime of the communication pattern.

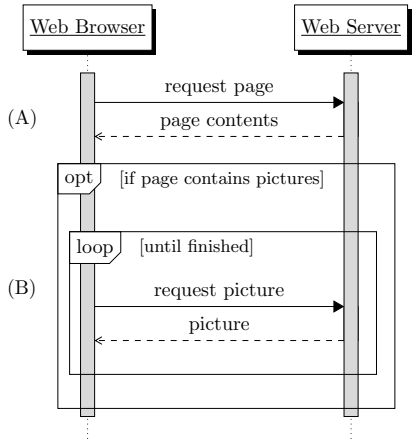


Fig. 8. Communication pattern example: Web browser requesting a web page which may contain pictures.

Messages can be synchronous (also termed calls) or asynchronous. A call consists of the two messages request and reply. An example of the synchronous call consisting of the request *request page* (filled arrow) and the reply *page contents* (non-filled arrow and dashed line) can be seen in fig. 8 step (A). Asynchronous messages consist of a request without any reply. Those are drawn with a non filled arrow.

An additional component is a *combined fragment*, which consists of an *operator kind*, a *constraint*, and an *operand* [104]. Combined fragments are drawn with a box enclosing the operand, which is an arbitrary combination of messages or further fragments. The upper left of the box contains the operator kind which specifies the type of fragment and optionally a constraint enclosed by square brackets. An example of a combined fragment can be seen in fig. 8 step (B).

Combined fragments used in this work are *opt* and *loop*. Fragments with the operator kind *opt* denote optional parts of the communication pattern [104]. The contents of this fragment occur only if the constraint is met. For example, fig. 8 step (B) is applicable only if the page contains pictures. The *loop* fragment is used to depict repeating communication [104]. For every picture the *request picture* call is executed in step (B). The *opt* fragment in fig. 8 contains only the *loop* fragment. Therefore, step (B) could have also been depicted with only the *loop* fragment since it gets executed for every picture, which is zero times in the case of no pictures.

Another UML sequence diagram part used in this work is an *interaction use*. This part is a reference to another sequence diagram. An interaction use is drawn like a combined fragment with *ref* as the operator kind and without a constraint [104]. In this work those are used to refer to parts that can not be generalized, like the infection part. Examples can be seen in fig. 9. If messages are exchanged through the network during such a reference, this is additionally depicted by an arrow in this work (e.g., fig. 9a). This is not part of the UML standard [105] and used as a visual aid.

The following communication patterns use a generalized list of common messages. These messages and calls are explained in the following list:

coordinate: Coordination message. In case the specific task

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

is not automated, this is needed to instruct the bot what to do.

scan: Network scan. This could be an ICMP echo request, a TCP or UDP scan, or a targeted scan for vulnerable services.

data: General data. This could be a bot binary, arbitrary files, or network data that is communicated with the botnet or other hosts.

register: Message needed for rallying. In case of a P2P topology this is needed to set up peer to peer communication.

execute, gather/compute, install/store, infect: Generalized tasks which are executed by the bot net. Those are either not visible in the network or specific to the botnet.

B. Overview

Depending on the topology, C&C traffic arriving at a bot can either be sent by another bot or by the C&C server. To keep the sequence diagrams simple, the generic term C&C is used as the communication partner of the bot. In a centralized topology C&C is one of the C&C servers, in the P2P topology another bot, and if proxying is used a proxy. The aim of this work is to present generalized communication patterns related to bot detection, which is why the distinction between C&C server and bot is not relevant. Commands could also be relayed through another bot instead of originating from the C&C server directly.

One important part of botnet communication is coordination. Except for automated tasks, every task is executed only after a command is received via the C&C channel. Without this coordination, only the effective botnet actions can be monitored by a third party. Therefore, the more tasks are automated, the stealthier the botnet is. Examples for automated tasks in botnets include the harvesting of login information in *Sality* [37] and *Storm* [93], as well as self-updating in *Stuxnet* [5], and network traffic modification in *Miner* [42] and *Zeroaccess* [98]. Limiting coordination comes at the expense of flexibility. Hence, there are only a limited number of tasks that are automated in current botnets, excluding highly specialized ones which were built for a dedicated purpose (e.g., *Stuxnet* [5]). One exception to this is the *Conficker* botnet, which was only capable of distributing and executing new binaries [84]. Since this process was initiated by the bots themselves, every task in the botnet was automated.

Communication in botnets can happen via the pull method or the push method. Pulling means that a bot requests outstanding commands from C&C, while the push method means that C&C sends commands to the bots actively. The used method depends mainly on the protocol in use. E.g., HTTP is built upon a request-response model, which means bots are only capable of using the pull method (see section IV-B for a more detailed explanation of HTTP). Additionally, the transport protocol might need acknowledgments which can be seen as data flowing into the other direction. To keep the following interaction diagrams simple, the distinction between push and pull, and possible acknowledgments are left out. In a network-based detector this distinction can also be left out by including a filter step that converts pull-based communication to push-based communication. Including this step is necessary to become protocol independent.

Botnet communication can be separated into the two stages *propagation* and *operation*. From the view of the infected host and bot binary these two steps can only occur sequentially since the infection (propagation) has to occur before the bot can become operational.

The first stage is propagation, which is used for recruiting new bots. Propagation can happen actively, when the botnet tries to infect additional victims, or passively, when the bot binary is distributed via other means (e.g., email or drive-by download). After propagation, newly created bots may register with the botnet. If the propagation happens passively and the registration is omitted, then this step can only be observed with Deep Packet Inspection (DPI). With DPI it is possible to extract binaries from network traffic which could then be classified as bot binaries. This approach works only if no encryption is used. Section VI-C discusses propagation methods in more detail.

The second stage is the operation stage. During this stage the botnet performs the actual work. Network communication during the operation stage depends on the purpose of the botnet. Section VI-D explains the different operation modes of a botnet in detail.

C. Propagation

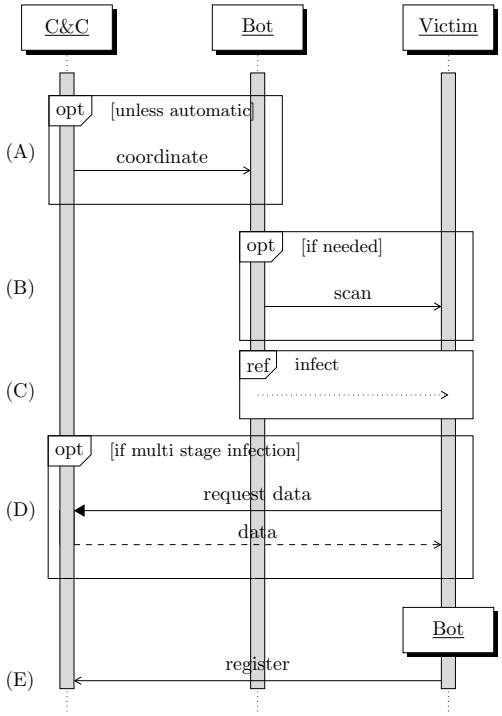
Propagation is the first step in the life of a bot. Since the potential power of a botnet increases with its size, botnets continuously try to recruit new bots. E.g., botnets capable of DDoS attacks can cause more traffic with more bots, botnets used to steal data can accumulate more data by spreading to additional hosts, and the spam volume of botnets sending email spam is directly dependent on the number of bots. APTs on the other hand need to stay as hidden as possible. Additionally, APTs are targeted attacks, and therefore only a specific number of infected machines is needed. These botnets exhibit the propagation step only seldom.

The propagation can happen either *actively* or *passively*, and is followed by a *registration step*.

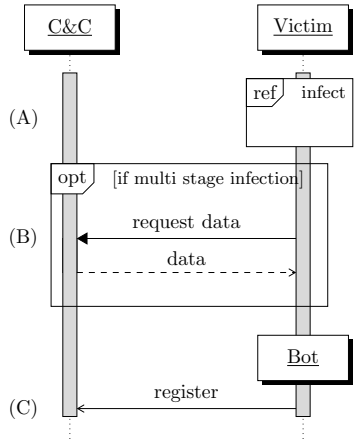
Both propagation techniques infect a victim host with the botnet executable and launch the bot. This infection can happen in multiple stages where the victim first runs only a small bootstrap binary which in turn fetches the actual bot executable from the botnet or even a completely different botnet (see fig. 9a step (D) and fig. 9b step (B)). For example, the *Rustock* botnet used a multistage installer where the last stage was encrypted by the C&C server with a key generated from system information of the target computer [90]. This technique increases the difficulty for researchers to obtain a bot executable for reverse engineering.

1) Active: Propagation through bots that use existing vulnerabilities for infecting additional hosts is called active propagation. This can happen on command or automatically. Depending on the implementation of the used exploit, a scanning step might be necessary before the actual infection can occur. This might be used to keep the needed computational resources at a minimum since, depending on the attack, scanning can

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/



(a) Active: Bots infect new machines via network with an optional scan step followed by the actual exploit.



(b) Passive: The victim is compromised indirectly.

Fig. 9. Propagation without communication infrastructure: Infection, optionally downloading additional data, and rallying (register).

be more lightweight. Another option is to use scanning to decrease the visibility of the attack if the actual infection uses protocols which are used seldom in the target network.

The complete communication pattern can be seen in fig. 9a. The first step (A) is a coordination step. Coordination is used to configure scan and exploit parameters. Possible parameters include which exploit or scan to use, limits on the targeted hosts, or botnet coordination to prevent multiple bots from approaching the same target. Finally, the coordination step instructs the bot to start the infection. Some botnets have the ability to store the mentioned parameters in a configuration file which is deployed alongside the bot or hard code the parameters in the bot binary. These botnets do not need a coordination step, and therefore step (A) in fig. 9a is optional.

The next step (B) is scanning. This step is optional since not every botnet scans the network before the actual exploit is tried, relying on pre-knowledge of victims instead. Scanning can consist of detecting reachable hosts with ICMP echo requests or directly looking for vulnerable services with UDP or TCP port scans. Botnets might opt to directly use the infection step for scanning where the payload is tried to be executed on a list of hosts.

The main step (C) is the actual infection. During this step vulnerabilities in software running on the victim host are exploited. Since this step depends on the actual exploit used, it is depicted as a reference. This step can be for instance a request containing a malicious payload or trying to get access to a service by guessing passwords. Since the maximum size of bot binaries which can be transmitted through the network during this step can be limited, an additional data download (step (D)) might be needed.

The data download step, which is part of a so called multi-stage infection, can also be used to complicate reverse engineering the bot binary. In this case only a small dropper is executed on the victim host. The dropper then downloads the actual bot binary, optionally decrypts it, sets up the needed environment, and executes it.

The last step (E) is the registering step. This step can be used to keep track of the botnet infections or might be needed to maintain the botnet topology. A more detailed description can be found in section VI-C3.

Examples for botnets that automatically infect additional computers are *Conficker* [30], *Sality* [37], *Slapper* [63], and *Stuxnet* [5]. *Phatbot* can use a hard coded configuration or can be instructed to scan specific network ranges [53]. Since scanning causes additional network traffic, this step is omitted in botnets like *Sality* [37].

2) Passive: Botnets can also be distributed via other vectors that are not in control of the botnet as detailed in fig. 9b. This is called passive propagation and includes propagation via emails, websites, or storage media. Common to these passive mechanisms is that users infect the victim host themselves by a click or an action.

Distribution via email can be performed using so-called phishing emails. A phishing email is an email that looks like a legitimate one, but is used to trick the receiving user into either opening an attached file or a compromised URL. Analogous to email infections there are also phishing websites that look like legitimate ones. One way to lure an unsuspecting target at a phishing website is to implement a copy of a

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

TABLE III
BOTNET PROPAGATION SUMMARY

Botnet	Active	Passive	Coordination	Scanning	Registration
Adwind	–	✓	–	–	✓
Blackenergy	–/✓[a]	✓	–/✓[a]	–/✓[a]	✓
Conficker	✓	✓	–	✓	–
Duqu 2.0	–/✓[a]	✓	–/✓[a]	–/✓[a]	–/✓[b]
Miner	–	✓	–	–	✓
Phatbot	✓	✓	✓	✓	✓
Regin	–/✓[a]	[c]	[c]	[c]	[c]
Rustock	–	✓	–	–	✓
Sality	✓	✓	–	–	✓
Sinit	–	✓	–	–	✓
Slapper	✓	–	–	✓	✓
Storm	–	✓	–	–	✓
Stuxnet	✓	✓	–	✓	✓[d]
TFN[e]	–	✓	–	–	–[f]
Trinoo	–	✓	–	–	✓
Waledac	–	✓	–	–	✓
Zeroaccess	–	✓	–	–	✓
Zeus	–	✓	–	–	✓

✓ Featured. – Not Featured.
[a] Only for lateral movement in network.
[b] Infected machines without pre-programmed C&C server stay dormant until woken [52].
[c] Unknown [50], [88].
[d] Only if Internet connection is available [5].
[e] Tribe Flood Network.
[f] List of bot IPs needs to be maintained manually.

website with common mistypes in the web address. Instead of involving the user, it is also possible to use vulnerabilities in the client browser or plugins therein. In this case, the botnet gets installed automatically upon visit. This is called a drive-by download. It can be used by infecting a website which is used regularly by the targeted victims. Targeting the attack this way is called a watering hole attack. Another way is to infect an advertising (Ad) network, which provides Ads to legitimate websites.

Only the underlying communication, like receiving an email or browsing a website, can be detected by a network-based detector. Since the user might have to actively open a downloaded file, the time of infection can be unrelated to the time of download. Additionally, the communication is the same as for non malicious emails or web sites. Therefore, a network-based detector can only detect the malicious content by employing DPI, which is made impossible with the use of encryption.

Removable storage media can be used to breach air gaps. Air gaps are computers or parts of a network that are not connected to the rest of the network for security reasons. If the complete botnet binary resides on the removable storage medium and the storage medium is used for communication, this form of propagation can not be detected by network-based botnet detection since no network traffic is exchanged.

The communication pattern which occurs during passive propagation can be seen in fig. 9b. First the bot binary is executed on the victim in step (A). Like with active propagation, this first binary could also be only a part of the bot executable. Therefore, the next step (B) is downloading additional data in case of a multi step infection. A more detailed explanation of this step can be seen in section VI-C1. The last step (C) is the registration with the botnet, which will be explained in section VI-C3.

Targeted phishing emails which installed the *Adwind* botnet, were used in attacks against different banking institutions [2]. The *Waledac* botnet distributes itself by sending high volume of infected spam [32]. In addition to email, the *Zeus* botnet is also distributed via drive-by downloads [99]. *Stuxnet* employs removable storage media in addition to active propagation to breach air gaps [5], [106]. The *Conficker* botnet was also capable of infecting removable storage media to further increase its reach [84].

3) Registration: The last step of the propagation process is the registration (see fig. 9b step (C) and fig. 9a step (E)). This step is also called rallying [9]. Registration adds the possibility for monitoring the botnet status, size, and is also needed for push-based setups since the server needs to know where to send the commands to. In P2P topologies this step can also be used to obtain a list of bots which is needed for bootstrapping the P2P network.

The *Adwind* botnet used this step to verify the software subscription status of the botmaster [2]. One of the earliest botnets, the *Tribe Flood Network*, left out the registration step, and therefore the list of bots had to be maintained manually [60], [95]. This was automated by the later incarnation *Stacheldraht*, which combined the source code of the *Tribe Flood Network* and other botnets [60].

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

4) Propagation Summary: The first step in the life cycle of a bot is propagation. In this section the communication patterns needed for the two propagation types *active* and *passive* are presented. Both start with the infection of the victim. Since during the active propagation a bot infects a victim host, this type of propagation has an optional scanning and optional coordination step before the infection. After the successful infection, both propagation types conclude with the optional registering of the bot with the botnet.

A summary of the different types of propagation and the registration used in botnets is presented in table III. This table lists for every analyzed botnet whether propagation occurs actively, passively, or via both methods. Additionally, the optional steps coordination, scanning, and registration are listed.

D. Operation

After propagation, a bot executes its assigned tasks. During this stage, called operation, various communication needs arise depending on the specific objectives. An example of these communication needs is coordinating DDoS so that every bot attacks the same target at the same time. A description of how DDoS attacks work can be seen in section IV-G. Another example are data harvesting bots, which need to report the harvested data back to the C&C or to the botmaster.

The needed control communication varies in the amount of transmitted data, direction of data flow, and the communication pattern. Since exact amount and direction of data is dependent on the protocol used, this work focuses on the communication pattern. A communication pattern is a sequence of messages, which have to be exchanged in the correct order to reach a specific outcome. A more detailed description can be found at the beginning of this section. Depending on the used communication pattern, the operation can be categorized as i) data upload, ii) data download, iii) forward proxy, iv) reverse proxy, and v) instruction.

Data upload is used to upload generated or gathered data from the bots, while *data download* can be used to store data (e.g., additional malware) on the bots. The terms *upload* and *download* are viewed from the bot perspective.

Forward proxy utilizes the botnet to hide the real origin of communication. This can be used for privacy reasons or to amplify traffic.

A *reverse proxy* is capable of relaying requests coming in from the Internet to a specific origin host. With this technique it is possible to build a CDN for hiding the true origin, and add resilience against take-down or host failure.

An *instruction* is the communication pattern that requires the lowest amount of data transfer for completion since only a command has to be sent over the network. It is used in scenarios where bots execute specific tasks on the host computer (e.g., change configuration or delete a specific file).

The various communication pattern variants are discussed in detail in the list below.

A single botnet can have more than one operation mode. There are several botnets, including *Blackenergy* [81], *Duqu 2.0* [52], and *Regin* [50], that are built modular. Those botnets

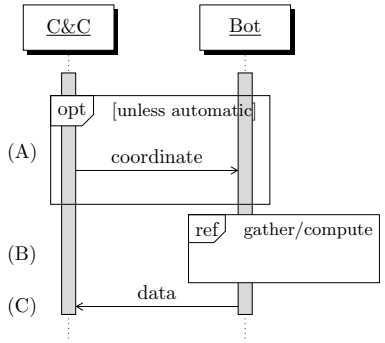


Fig. 10. Data upload: Gathered or computed data is uploaded from bot to botmaster.

can be preconfigured to execute certain tasks. Additionally, they allow dynamically changing the tasks that the botnet is capable of.

1) Data Upload: Botnets belonging to the categories *information gathering* and *distributed computing* defined by [9] use the data upload communication pattern. One example botnet task belonging to this category is uploading various types of information (e.g., login credentials, system information, arbitrary files) from the infected computers. Another possible task is to use the botnet for computing data and then uploading the results (e.g., bitcoin mining, password cracking).

Data upload starts with an optional coordination (step (A) in fig. 10). During this step the C&C sends commands to the bots with specific instructions or data needed for the computation. Since the data upload task can happen automatically, e.g., directly after propagation, the coordination step is optional.

Next, the requested data is harvested from the computers or the requested computations are executed in step (B). Requested data can include for example email addresses, login credentials, software license keys, or keyboard logging data.

The last step (C) of the data upload communication pattern is to report the results back to C&C. Since this step depends on the duration of the assigned task, it can happen at a later time.

A botnet using data upload without coordination is *Waledac*. After infection, it automatically searches the infected computer for email addresses and login credentials, which are sent to C&C after discovery [32]. Another example for coordination-less data upload is the *Miner* botnet. Following propagation, it measures network performance, collects system information, and sends this data to the C&C automatically [42]. Botnets that use coordination for data upload include *Blackenergy* [81] and *Duqu 2.0* [52], which are capable of uploading various types of information from the infected computers. Two botnets that use infected computers for computation are *Miner* [42] and early versions of *Zeroaccess* [41]. Both include a bit coin miner, which utilizes the computational power of the bots to harvest money on behalf of the botmaster.

A complete list of botnets that are analyzed by this work, along with their requested tasks and optional coordination activities is summarized in table IV. This table includes harvested data, such as *Logins* from *files* on the hard disc and *network traffic*, *information about* the *system*, connected *de-*

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

TABLE IV
DATA UPLOAD OVERVIEW

	Collects logins from		Collects information about			Captured events and data				Comp.		
Botnet	File	Net	Sys	Dev	Net	Traffic	AV[a]	Screen	Keyb.	Bitcoin	EmailA.	Generic
Adwind	C	–	N	–	–	–	C	C	C/N[b]	–	–	C
Blackenergy	C	C	C	C	–	C	–	C	C	–	–	C
Conficker	–	–	–	–	–	–	–	–	–	–	–	–
Duqu 2.0	C	C	C	C	C	–	–	C	C	–	–	C
Miner	–	N	N	–	N	–	–	–	–	C	–	C
Phatbot	C	C	C	–	C	C	–	–	C	–	C	–
Regin	C	C	C	C	C	C	–	C	C	–	–	C
Rustock	–	–	N	–	–	–	–	–	–	–	–	–
Sality	N	–	–	–	–	–	–	–	–	–	–	–
Sinit	–	–	–	–	–	–	–	–	–	–	–	–
Slapper	–	–	–	–	–	–	–	–	–	–	–	–
Storm	N	–	–	–	–	–	–	–	–	–	N	–
Stuxnet	–	–	N	–	–	–	–	–	–	–	–	–
TFN[c]	–	–	–	–	–	–	–	–	–	–	–	–
Trinoo	–	–	–	–	–	–	–	–	–	–	–	–
Waledac	N	–	N	–	N	–	–	–	–	–	N	–
Zeroaccess	N	N	N	–	–	N	–	–	–	C[d]	–	–
Zeus	C/N	C/N	N	–	–	C/N	–	C/N	C/N	–	–	–

C Coordination. N No Coordination. – Not Featured.
[a] Microphone, Camera.
[b] Online and offline keylogging possible [2].
[c] Tribe Flood Network.
[d] Earlier versions. Has been removed from the botnet.

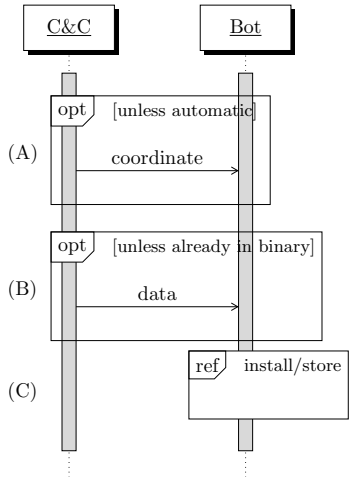


Fig. 11. Data download: Store or install data downloaded from adversary onto bot.

vices and *network*, and *email addresses*. Additionally, *captured* data from *network*, *audio or video* devices, and *screen-* and *keyboard*-sniffers are listed. Furthermore, *bitcoin computing* botnets and botnets that allow uploading of *generic* data are listed. Botnets which use coordination are marked with *C* and botnets that automatically upload data with *N*.

2) Data Download: This mode of operation is needed for storing data onto bots. This data can be arbitrary files, additional software (spreading malware in [9]), or a botnet binary update. Arbitrary files may include illegal content which can be used to discredit the victim. Additionally, this mechanism can be used to monetize the botnet by offering the bots as a potential installation base to other malware developers.

Adding the capability for self-update enables the botmaster to adapt the botnet to novel needs or to secure against future threats. This self-updating adds an attack vector to the botnet, which can be used to take over the botnet or replace it with a different one. Therefore, this procedure needs to be secured.

The data download communication pattern (see fig. 11) starts with the coordination step (A). During coordination the C&C instructs the bot which file should be downloaded and where to find it. This simplifies the C&C protocol since no file transfer capabilities are needed and expands the capabilities of the botnet allowing downloading of arbitrary files. Bot binaries can contain an additional configuration file which starts the data download right after propagation. Therefore, the coordination step is optional.

After coordination, the data has to be downloaded to the bot in step (B). This additional data can already be included in the bot binary used during propagation or be part of the coordination command.

The last step (C) is to store the downloaded files on the computer or install the downloaded software. Although there is no need for communicating back to C&C, the additional installed software or updates to the botnet client can cause additional traffic after this step. One example for this additional communication can be registration during this propagation (see section VI-C).

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

TABLE V
DATA DOWNLOAD OVERVIEW

	Install Software		
Botnet	Self-update	Additional	Arbitrary files
Adwind	–	C	C
Blackenergy	C	C	C
Conficker	N	N	–
Duqu 2.0	–	C	C
Miner	C/N[a]	C/N[a]	–
Phatbot	C	–	C
Regin	–	C	C
Rustock	C	–	–
Sality	–	C	–
Sinit	–	C	–
Slapper	–	–	–
Storm	C	C	–
Stuxnet	C	C	–
TFN[b]	–	–	–
Trinoo	–	–	–
Waledac	C	C	–
Zeroaccess	C	C	–
Zeus	C	C	C

C Coordination. N No Coordination. – Not Featured.
[a] Automatically after infection [42].
[b] Tribe Flood Network.

Installing additional malware was one of the main goals of the botnets *Conficker* [30], *Sality* [37], *Sinit* [39], and *Waledac* [32]. The *Miner* botnet can include additional software in the bot binary used during propagation, and therefore the coordination step is not used every time [42].

A complete list of botnets featured in this paper, and data download targets can be seen in table V. The targets include the *installation of software* in order to *self-update* the bot or install *additional* software. Some botnets are also capable of storing *arbitrary files* on the computer. Botnets that use coordination are marked with a *C*, while botnets that automatically download files to the victim are marked with *N*.

3) Forward Proxy: Botnets, which are used for distributing spam or performing DDoS attacks need the capability to connect to other hosts on behalf of the botmaster (see section IV-G). This mimics the botnet being used as a forward proxy. A forward proxy is an intermediate server that is able to make connections to specific hosts on behalf of the requesting host. Therefore, forward proxies can be used to preserve the anonymity of the requesting host, to tunnel connections through a different protocol, or to make connections which would be otherwise prohibited or impossible.

Botnets can implement either a *bidirectional* or a *unidirectional* forward proxy.

Bidirectional is used if the botmaster or a third party needs the result of a request. For example, some botnets include a Socket Secure (SOCKS) proxy service, which allows using the botnet as an anonymizing service. A SOCKS proxy is a generic proxy that can be used to forward arbitrary TCP and UDP connections. After an initial setup phase, the SOCKS proxy forwards raw data transparently, which allows the use of the proxy even with applications that do not support proxies [107].

Another example for bidirectional proxying is to traverse firewalls or NATs. This can be achieved by infecting the firewall or NAT device with a bot which in turn acts as a bridge into the local network.

To increase stealthiness of the botnet, forward proxying can also be used to hide protocols in one or several other protocols, i.e., to employ protocol encapsulation. With this approach a single C&C communication pattern can propagate over several protocols along the path. This technique increases the complexity of network-based botnet detectors since the detectors need to be capable of understanding all the implemented protocols and reconstruct the original communication.

Unidirectional forward proxying can be used to overload a single service with requests. This can be achieved by instructing bots to send requests to the network services continuously. Since neither C&C needs to control each message, nor are replies of interest, it suffices to instruct bots about the target and communication type.

Another way of employing unidirectional forward proxying is to use it for distributed email spam. During coordination spam templates and lists of destination addresses are sent to the bot. The list of destination addresses could originate from the botnet itself, where it was obtained via data upload (see section VI-D1). After the coordination step, the emails are distributed to email servers.

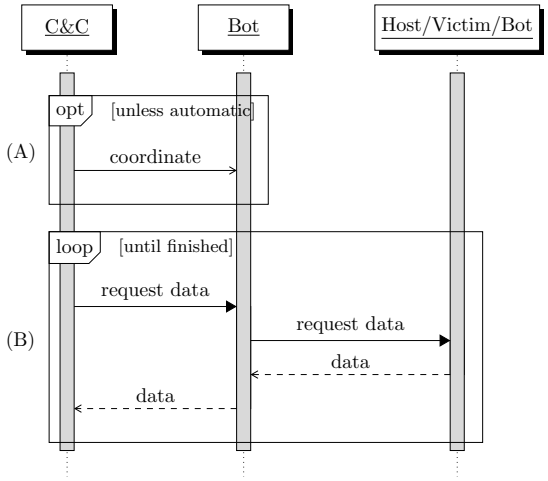
A variation of the same method can be used to perform click fraud (see also section IV-G). Like email spam, the target URLs are distributed to the bots during the coordination step, which in turn send the requests afterwards.

Botnets supporting these tasks are part of the categories called *cyber fraud*, *unsolicited marketing*, and *network service disruption* as defined in [9].

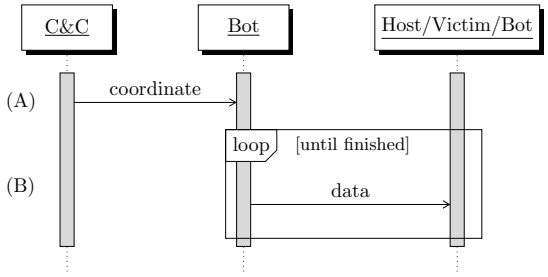
The communication pattern for the forward proxy mode can be seen in fig. 12. Both bidirectional and unidirectional start with coordination (step (A) in fig. 12a and step (A) in fig. 12b). In the bidirectional case the coordination step is used to set the proxy configuration including forwarded ports and protocols. Since this configuration can already be hard-coded in the bot binary, this step is optional. Since unidirectional proxying is used exclusively for handling automated tasks, the coordination step is not optional. E.g., for a DDoS attack during coordination the target, type of attack, and optionally an attack duration must be set up. Theoretically, this information could also be hard-coded in the bot binary, but the inflexibility limits the practicability of this approach due to short-lived and sudden nature of DDoS attacks. Therefore, all the analyzed botnets use the coordination step for unidirectional proxying.

The last step in the forward proxy communication pattern is the exchange of the data requests. Bidirectional communication consists of requests and replies that are relayed through the bot in step (B) in fig. 12a. In case the botnet is used as a generic proxy, the initiator of the requested data can also be a different host instead of C&C. Unidirectional forward proxying consists only of data sent from the bot to the target in step (B) in fig. 12b.

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/



(a) Bidirectional: Response is sent back.



(b) Unidirectional: Responses are ignored.

Fig. 12. Forward proxy: Relay communication originating from adversary via the botnet.

TABLE VI
FORWARD PROXY OVERVIEW

	Bidirectional					Unidirectional		
Botnet	Generic[a]	Port scan	Net scan	NAT trav.	Prot. transl.	Spam	DDoS	Click fraud
Adwind	C	–	–	–	–	–	–	–
Blackenergy	C	C	–	–	–	–	C	–
Conficker	–	–	–	–	–	–	–	–
Duqu 2.0	–	C	C	C	C	–	–	–
Miner	N	–	–	–	–	–	C	–
Phatbot	C	–	C	–	–	–	C	–
Regin	–	–	C	C	C	–	–	–
Rustock	–	–	–	–	–	C	–	–
Sality	–	–	–	–	–	–	–	–
Sinit	–	–	–	–	–	–	–	–
Slapper	–	–	–	–	–	–	C	–
Storm	–	–	–	–	–	C	C[b]	–
Stuxnet	–	–	–	–	–	–	–	–
TFN[c]	–	–	–	–	–	–	C	–
Trinoo	–	–	–	–	–	–	C	–
Waledac	–	–	–	–	–	C	–	–
Zeroaccess	–	–	–	–	–	–	–	C
Zeus	–	–	–	–	–	–	–	–

C Coordination. N No Coordination. – Not Featured.
[a] E.g., SOCKS-, HTTP-, or HTTPS-proxy.
[b] Launched against networks that appear to investigate the botnet [94].
[c] Tribe Flood Network.

Examples for bidirectional forward proxying botnets include *Miner* [42], *Adwind* [2], and *Phatbot* [38]. Those feature a SOCKS proxy in order to provide an anonymizing service. *Duqu 2.0* employs various proxy capabilities to enable NAT traversal and protocol translation in order to increase the stealthiness of the botnet [52].

The two botnets *Trinoo* and *Tribe Flood Network* use unidirectional forward proxying to perform DDoS attacks [60], [62]. An example for a botnet capable of distributing spam is *Rustock* [91]. *Zeroaccess* uses unidirectional forward proxy to implement click fraud [98].

A complete summary of the analyzed botnets can be seen in table VI. This summary includes tasks ordered by *bidirectional* mode and *unidirectional* mode. *Generic* stands for generic proxies such as SOCKS, HTTP, or HTTPS. *Port scan* and *network scan* depicts bots which are able to perform scanning on behalf of the botmaster. Botnets capable of *NAT traversal* and *protocol translation* use those techniques to increase the stealthiness and reachability of the C&C channel. Unidirectional proxy modes listed include the distribution of *spam*, *DDoS* attacks, and *click fraud*. Botnets capable of using this feature with coordination are marked with a *C*, whereas botnets

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

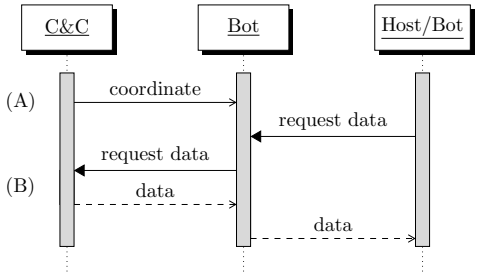


Fig. 13. Reverse proxy: Relay communication originating from the Internet via the botnet to the adversary.

that automatically launch a proxy after start up are marked with *N*.

4) Reverse Proxy: Botnets can be used to build a fast-flux network. A fast-flux network consists of one or more origin servers and reverse proxy servers that are used to connect to the origin server. Since connections are created to the proxy servers, the identity of the origin is hidden. This makes the origin server resilient against take down. This is covered in more detail in section III-A.

The network communication needed for reverse proxy starts with the coordination phase (step (A) in fig. 13). During coordination, C&C instructs the bot which port to open and which internal address to connect to. After coordination the reverse proxy can be used by other hosts or bots to connect to the origin (step (B)). Due to the dynamic nature of botnets caused by hosts that are not online all the time, disinfections, and network failures it is unfeasible to hard-code this information. Therefore, coordination was used in every analyzed bot implementing a reverse proxy.

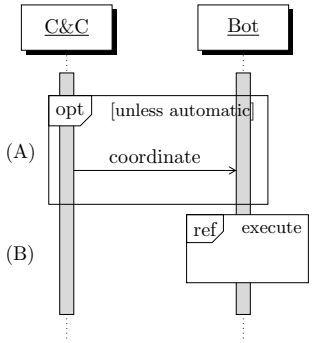
Only three of the analyzed botnets employ fast-flux functionality: *Waledac* [96], *Storm* [93], and *Zeus* [43].

5) Instruction: A feature present in botnets is the capability to execute tasks on behalf of the botmaster which do not cause network traffic. These instructions can affect either the computer hosting the bot or a connected peripheral.

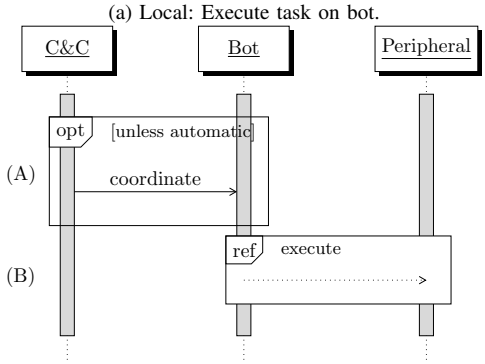
Example tasks involving the host include terminating other processes, deleting files, modifying host network traffic, disrupting services, or providing general functionality to control every aspect of the host (backdoor).

Connected peripherals are connected via a non network connection, via a separate network that can not be monitored by the network detector, or on the same network as the detector. Examples for such devices are SCADA peripherals like Programmable Logic Controller (PLC). A PLC is a system capable of controlling processes in industrial machines.

The communication pattern for both types of instruction can be seen in fig. 14. The first step is coordination (step (A) in fig. 14a and step (A) in fig. 14b). During coordination the botmaster instructs the bots which tasks to execute and provides needed task parameters. Bot binaries can also be bundled with a pre created configuration containing such instructions or include hard coded instructions into the binary. For example, many botnets contain a list of anti virus processes to kill. In this case no coordination step is needed and instructions are executed right after propagation.



(a) Local: Execute task on bot.



(b) Remote: Execute task on connected peripheral.

Fig. 14. Instruction: Execute task.

After the coordination step the actual command is executed in step (B). This can either occur on the host of the bot or on a connected peripheral. Since connected peripherals can reside on an observable network, this step can potentially be seen by a network-based detector.

Botnets that can execute various commands (e.g., delete files, modify network traffic) on behalf of the botmaster on the host computer are *Blackenergy* [82], *Duqu 2.0* [52], and *Regin* [50]. Examples for hard coded instructions include *Conficker* [30], *Adwind* [2], *Miner* [42], *Sality* [37], and *Phatbot* [53], which contain a hard coded list of processes to terminate or disable after propagation. An example for a botnet that can execute tasks on connected peripherals is *Stuxnet*, which is capable of re-programming connected PLCs [5]. Although *Conficker* can only distribute and execute new binaries, bots can also be controlled via this mechanism. This works by loading the binaries in the process context, which gives the new binary full access to the bot [84].

An overview of all analyzed botnets and different instructions can be seen in table VII. This table contains the instructions for *exiting* and *configuring* the *bot* binary, as well as *terminating* foreign *processes*. Some botnets are also capable of *deleting files* or *check* for their existence. There are also botnets capable of *modifying* the network traffic on the host for spoofing websites or emulating network services. More disruptive capabilities include *destroying* or *encrypting* the *Hard Disc Drive (HDD)*, or disrupting connected peripherals like *SCADA* or *CISCO network devices*. Many botnets are also capable of providing a *generic* backdoor.

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

TABLE VII
SUMMARY OF SUPPORTED BOTNET INSTRUCTIONS

	Bot		Process	Files		Traffic	HDD		Disrupt		
Botnet	Exit	Config.	Term.	Delete	Check	Modify	Destroy	Encrypt	CISCO	SCADA	Generic[a]
Adwind	C	C	C/N	C	C	–	–	–	–	–	C
Blackenergy	C	C	C	C	C	C	C	C	C	C	C
Conficker	N	–	N	–	–	N	–	–	–	–	N[b]
Duqu 2.0	–	–	–	C	C	C	–	–	–	–	C
Miner	–	–	N	–	–	N	–	–	–	–	–
Phatbot	C	C	C/N	–	–	–	–	–	–	–	C
Regin	C	C	C	C	C	C	–	–	–	–	C
Rustock	–	–	C	C	–	–	–	–	–	–	–
Sality	–	–	N	N	–	–	–	–	–	–	–
Sinit	–	–	–	–	–	–	–	–	–	–	–
Slapper	–	–	–	–	–	–	–	–	–	–	–
Storm	–	–	–	–	–	–	–	–	–	–	–
Stuxnet	N	–	–	–	–	–	–	–	–	N	–
TFN[c]	–	–	–	–	–	–	–	–	–	–	C[d]
Trinoo	C	–	–	–	–	–	–	–	–	–	–
Waledac	–	–	–	–	–	–	–	–	–	–	–
Zeroaccess	–	–	–	–	–	N	–	–	–	–	C
Zeus	–	C	–	C[e]	–	–	–	–	–	–	C

C Coordination. N No Coordination. – Not Featured.
[a] Backdoor.
[b] Loads binaries into process context, therefore allowing dynamic reconfiguration.
[c] Tribe Flood Network.
[d] Optional.
[e] System files, rendering the computer unusable [99].

TABLE VIII
BOTNET COMMUNICATION SUMMARY

		Data		Proxy		
Botnet	Propagation	Upload	Download	Forward	Reverse	Instruction
Adwind	PR	C/N	C	C	–	C/N
Blackenergy	(ACS)PR	C	C	C	C	C
Conficker	APS	–	N	–	–	N
Duqu 2.0	(ACS)P(R)	C	C	C	C	C
Miner	PR	C/N	C/N	C	–	N
Phatbot	ACSPR	C	C	C	C	C/N
Regin	(A)[a]	C	C	C	C	C
Rustock	PR	N	C	C	–	C
Sality	APR	N	C	–	–	N
Sinit	PR	–	C	–	–	–
Slapper	AR	–	–	C	–	–
Storm	PR	N	C	C	C	–
Stuxnet	APSR	N	C	–	–	N
TFN[b]	P	–	–	C	–	C[c]
Trinoo	PR	–	–	C	–	C
Waledac	PR	N	C	C	C	–
Zeroaccess	PR	N	C	C	–	C/N
Zeus	PR	C/N	C	–	–	C

A Active. C Coordination. P Passive. S Scanning. R Registration. N No Coordination. – Not Featured. () Only for lateral movement.
[a] Unknown [50], [88]. [b] Tribe Flood Network. [c] Optional.

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

E. Operation Summary

After propagation, the botnet performs the actual tasks during operation. Tasks from a diverse set of botnets varying in sophistication, age, purpose, and topology have been analyzed with respect to network communication. The botnet tasks from this list were categorized according to their objective and communication pattern into: i) data download, ii) data upload, iii) forward proxy, iv) reverse proxy, and v) instruction.

These fundamental communication patterns can be used to match botnet behavior and build the base for a network-based botnet detector.

A summary of all the communication patterns is shown in table VIII. This table lists the botnets analyzed in this work along with their communication patterns and whether coordination (*C*) is involved. The propagation column lists how propagation is performed with respect to active or passive propagation, scanning, and registration. Propagation modes in brackets are only used for lateral movement, which is propagation in the local network.

VII. Conclusion

The botnet landscape is changing over time. Driven by the objective to hide within legitimate systems and communications, botnets follow the evolution of applications and protocols in the Internet. With respect to communication protocols, recent botnet implementations prefer HTTP and proprietary protocols over the IRC protocol, that was predominant in early botnets. Concurrently, botnet topologies evolved from a centralized concept to more distributed and resilient control structures. Common to many recent botnets is the shifting of their focus toward stealthiness and encryption in order to increase their expected lifetime until detection and take-down. But despite these trends centralized botnets still exist today and even some newly created ones rely on legacy techniques to keep complexity and implementation effort at a low level. Summarizing, the increasing use of encryption and the wide diversity in active botnet variants are two main reasons why detecting botnets via network-based methods is an essential, rewarding, but also challenging research field for now and in the future.

In this paper we have presented an in-depth analysis of botnet communication. The first part of this paper consists of botnet fundamentals and network-based detectors in section I and an extended survey of botnet topologies in section III. The topologies and accompanying challenges are discussed from a network communication perspective including examples of existing botnets and a historic context.

This is followed by a detailed list of C&C protocols that are known and in use as of writing this paper in section IV. Protocol details include how the protocol works, possible botnet topologies and discussion how to implement those, hiding and obfuscation techniques, and example botnets.

The main part of this paper is a novel taxonomy of botnet C&C network communication patterns which was derived from a diverse set of botnets described in section V. Botnets in this set were chosen to represent a distinct list varying according to age, used protocols, used topology, and sophistication.

Following this description we present the communication patterns subdivided into the generalized scenarios i) propagation, ii) data download, iii) data upload, iv) forward proxy, v) reverse proxy, and vi) instruction in section VI. The scenarios include an introductory description helping to understand why they are needed, a description of the message exchanges, and examples of botnets using those. These communication patterns and scenarios cover every possible botnet task, which can also be seen in the accompanying tables providing an overview of the analyzed botnets with respect to the scenarios. Additionally, visualizations are provided in standardized UML sequence diagrams, which ease the implementation of a possible network-based botnet detector and can be easily extended in the future.

The generality and extensibility of the proposed communication pattern abstractions recommends them as indispensable mechanism to support researchers in developing advanced detection tools. Examples of possible applications include, but are not limited to, network-based botnet detectors or event generation engines as mentioned in [20].

A hypothetical botnet detector could use machine learning to extract message exchanges from network traffic. These message exchanges can then be tested against the proposed dialogs in section VI, identifying possible botnet communication.

The main benefit of a detector capable of matching communication patterns is its independence from C&C protocol, topology, and botnet family. Additionally, depending on the way those patterns are matched, the detection of encrypted C&C protocols is also possible.

List of Acronyms

Ad advertising
AES Advanced Encryption Standard
APT Advanced Persistent Threat
ASCII American Standard Code for Information Interchange
C&C Command and Control
CDN Content Delivery Network
DDoS Distributed Denial of Service
DGA Domain Generation Algorithm
DNS Domain Name System
DPI Deep Packet Inspection
email electronic mail
FastLZ Fast Lempel Ziv
FTP File Transfer Protocol
HDD Hard Disc Drive
HTTP Hyper Text Transfer Protocol
HTTPS Hyper Text Transfer Protocol Secure
ICMP Internet Control Message Protocol
IDS Intrusion Detection System
IP Internet Protocol
IPC Inter-Process Communication
IRC Internet Relay Chat
JPEG Joint Photographic Experts Group
LZF Lempel Ziv Free
LZJB Lempel Ziv Jeff Bonwick
LZO Lempel Ziv Oberhumer
malware malicious software

This article is the accepted version of "Botnet Communication Patterns" in the journal "Communications Surveys & Tutorials".
Citation information: DOI 10.1109/COMST.2017.2749442 Abstract: http://ieeexplore.ieee.org/document/8026031/

MD5 Message Digest 5
NAT Network Address Translation
P2P Peer to Peer
PLC Programmable Logic Controller
PPI Pay-Per-Install
RAT Remote Administration Tool
RC4 Rivest Cipher 4
RC5 Rivest Cipher 5
RSA Rivest-Shamir-Adleman
SCADA Supervisory Control and Data Acquisition
SMB Server Message Block
SMTP Simple Mail Transfer Protocol
SOA Service Oriented Architecture
SOCKS Socket Secure
TCP Transport Control Protocol
TEA Tiny Encryption Algorithm
TLS Transport Layer Security
UDP User Datagram Protocol
UML Unified Modeling Language
URL Uniform Resource Locator
USB Universal Serial Bus
VPN Virtual Private Network
XOR exclusive or
XTEA Extended Tiny Encryption Algorithm

REFERENCES

[1] C. Wueest, "Financial threats 2015," Symantec Corporation, Tech. Rep. Version 1.0, Mar. 2016. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/financial-threats-15-en.pdf

[2] V. Kamluk and A. Gostev, "Adwind - A cross-platform RAT," Kaspersky lab, Tech. Rep., Feb. 2016. [Online]. Available: https://securelist.com/securelist/files/2016/02/KL_AdwindPublicReport_2016.pdf

[3] "ZeroAccess Botnet Resumes Click-Fraud Activity," SecureWorks, Jan. 2015. [Online]. Available: https://www.secureworks.com/blog/zeroaccess-botnet-resumes-click-fraud-activity-after-six-month-break

[4] O. Kupreev, J. Strohschneider, and A. Khalimonenko, "Kaspersky DDOS intelligence report for Q3 2016," Kaspersky lab, Tech. Rep., Oct. 2016. [Online]. Available: https://securelist.com/analysis/quarterly-malware-reports/76464/kaspersky-ddos-intelligence-report-for-q3-2016/

[5] N. Falliere, L. O Muruchu, and E. Chien, "W32.Stuxnet Dossier," Symantec Corporation, Tech. Rep. Version 1.4, Feb. 2011. [Online]. Available: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf

[6] B. Krebs, "KrebsOnSecurity Hit With Record DDoS," Sep. 2016. [Online]. Available: https://krebsonsecurity.com/2016/09/krebsonsecurity-hit-with-record-ddos/

[7] R. M. Lee, M. J. Assante, and T. Conway, "Analysis of the Cyber Attack on the Ukrainian Power Grid," Mar. 2016.

[8] D. Salmi, "Could an Internet of Things botnet army threaten the internet?" Oct. 2016. [Online]. Available: https://blog.avast.com/could-an-internet-of-things-botnet-army-threaten-the-internet

[9] S. Khattak, N. R. Ramay, K. R. Khan, A. A. Syed, and S. A. Khayam, "A Taxonomy of Botnet Behavior, Detection, and Defense," *IEEE Commun. Surv. Tutor.*, vol. 16, no. 2, pp. 898–924, Second Quarter 2014.

[10] W. Z. Khan, M. K. Khan, F. T. B. Muhaya, M. Y. Aalsalem, and H. C. Chao, "A Comprehensive Study of Email Spam Botnet Detection," *IEEE Commun. Surv. Tutor.*, vol. 17, no. 4, pp. 2271–2295, Fourth Quarter 2015.

[11] N. Hoque, D. K. Bhattacharyya, and J. K. Kalita, "Botnet in DDoS Attacks: Trends and Challenges," *IEEE Commun. Surv. Tutor.*, vol. 17, no. 4, pp. 2242–2270, Fourthquarter 2015.

[12] J. Livingood and N. Mody, "Recommendations for the Remediation of Bots in ISP Networks," RFC6561, Mar. 2012.

[13] R. W. Shirey, "Internet Security Glossary, Version 2," RFC 4949, Aug. 2007.

[14] S. Saad, I. Traore, A. Ghorbani, B. Sayed, D. Zhao, W. Lu, J. Felix, and P. Hakimian, "Detecting P2P botnets through network behavior analysis and machine learning," in *2011 Ninth Annual International Conference on Privacy, Security and Trust (PST)*, Jul. 2011, pp. 174–180.

[15] M. Alauthaman, N. Aslam, L. Zhang, R. Alasem, and M. A. Hossain, "A P2P Botnet detection scheme based on decision tree and adaptive multilayer neural networks," *Neural Comput & Applic*, pp. 1–14, Oct. 2016.

[16] L. Bilge, D. Balzarotti, W. Robertson, E. Kirda, and C. Kruegel, "Disclosure: Detecting Botnet Command and Control Servers Through Large-scale NetFlow Analysis," in *Proceedings of the 28th Annual Computer Security Applications Conference*, ser. ACSAC '12. New York, NY, USA: ACM, 2012, pp. 129–138.

[17] R. Sommer and V. Paxson, "Outside the Closed World: On Using Machine Learning for Network Intrusion Detection," in *2010 IEEE Symposium on Security and Privacy (SP)*, May 2010, pp. 305–316.

[18] S. García, "Survey on Network-based Botnet Detection Methods," *Secur. Commun. Netw.*, 2013.

[19] G. Gu, R. Perdisci, J. Zhang, and W. Lee, "BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection," ser. 2, vol. 5, 2008, pp. 139–154.

[20] A. B. Ashfaq, Z. Abaid, M. Ismail, M. U. Aslam, A. A. Syed, and S. A. Khayam, "Diagnosing bot infections using Bayesian inference," *J Comput Virol Hack Tech*, pp. 1–18, Sep. 2016.

[21] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, "BotHunter: Detecting Malware Infection Through IDS-driven Dialog Correlation," in *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, ser. SS'07. Berkeley, CA, USA: USENIX Association, 2007, pp. 12:1–12:16.

[22] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir, "A Survey of Botnet Technology and Defenses," in *Conference For Homeland Security, 2009. CATCH '09. Cybersecurity Applications Technology*, Mar. 2009, pp. 299–304.

[23] M. Eslahi, R. Salleh, and N. B. Anuar, "Bots and botnets: An overview of characteristics, detection and challenges," in *2012 IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, Nov. 2012, pp. 349–354.

[24] Z. Zhang, R. Ando, and Y. Kadobayashi, "Hardening Botnet by a Rational Botmaster," in *Information Security and Cryptology*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Dec. 2008, no. 5487, pp. 348–369.

[25] S. Yadav, A. K. K. Reddy, A. N. Reddy, and S. Ranjan, "Detecting Algorithmically Generated Malicious Domain Names," in *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement*, ser. IMC '10. ACM, 2010, pp. 48–61.

[26] D. Plohmann, K. Yakdan, M. Klatt, J. Bader, and E. Gerhards-Padilla, "A Comprehensive Measurement Study of Domain Generating Malware," in *25th USENIX Security Symposium*, Austin, TX, Aug. 2016.

[27] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," RFC 3986, Jan. 2005.

[28] P. Mockapetris, "DOMAIN NAMES - IMPLEMENTATION AND SPECIFICATION," RFC 1035, Nov. 1987.

[29] K. O'Meara, D. Shick, J. Spring, and E. Stoner, "Malware Capability Development Patterns Respond to Defenses: Two Case Studies," White Paper, Software Engineering Institute, Carnegie Mellon University, 2016.

[30] F. Leder and W. Tillmann, "Know Your Enemy: Containing Conficker," The Honeynet Project, Tech. Rep., Apr. 2009. [Online]. Available: https://www.honeynet.org/sites/default/files/files/KYE-Conficker.pdf

[31] J. Stewart, "Inside the Storm: Protocols and Encryption of the Storm Botnet," Blackhat Technical Security Conference, 2008.

[32] G. Tenebro, "W32.Waledac Threat Analysis," Symantec Corporation, Tech. Rep., 2009. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/W32_Waledac.pdf

[33] "TRANSMISSION CONTROL PROTOCOL," RFC 793, Sep. 1982.

[34] C. Rossow, D. Andriesse, T. Werner, B. Stone-Gross, D. Plohmann, C. Dietrich, and H. Bos, "SoK: P2PWNED - Modeling and Evaluating the Resilience of Peer-to-Peer Botnets," in *2013 IEEE Symposium on Security and Privacy (SP)*, May 2013, pp. 97–111.

[35] P. Wang, S. Sparks, and C. C. Zou, "An Advanced Hybrid Peer-to-Peer Botnet," *IEEE Trans. Dependable Secure Comput.*, vol. 7, no. 2, pp. 113–127, Apr. 2010.

[36] B. Stone-Gross, "The Lifecycle of Peer to Peer (Gameover) ZeuS," SecureWorks, Tech. Rep., Jul. 2012. [Online]. Available: https://www.secureworks.com/research/the_lifecycle_of_peer_to_peer_gameover_zeus

[37] N. Falliere, "Sality: Story of a Peer-to-Peer Viral Network," Symantec Corporation, Tech. Rep. Version 1.0, Jul. 2011. [Online]. Available: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/sality_peer_to_peer_viral_network.pdf

[38] J. Stewart. (2004, Mar.) Phatbot Trojan Analysis. SecureWorks. [Online]. Available: http://web.archive.org/web/20080917193007/http://www.secureworks.com/research/threats/phatbot/

[39] ——. (2003, Dec.) Sinit P2P Trojan Analysis. SecureWorks. [Online]. Available: http://web.archive.org/web/20080921095447/http://www.secureworks.com/research/threats/sinit/

[40] P. Porras, H. Saidi, and V. Yegneswaran, "Conficker C P2P Protocol and Implementation," Technical report, SRI International, Tech. Rep., 2009.

[41] J. Morris, "Cracking the Encrypted C&C Protocol of the ZeroAccess Botnet," VirusBulletin, Kindsight Security Labs, Dallas, 2012. [Online]. Available: https://www.virusbulletin.com/uploads/pdf/conference_slides/2012/Morris-VB2012.pdf

[42] D. Plohmann and E. Gerhards-Padilla, "Case study of the Miner Botnet," in *4th International Conference on Cyber Conflict (CYCON)*, Jun. 2012, pp. 1–16.

[43] D. Andriesse, C. Rossow, B. Stone-Gross, D. Plohmann, and H. Bos, "Highly resilient peer-to-peer botnets are here: An analysis of Gameover Zeus," in *8th International Conference on Malicious and Unwanted Software: "The Americas" (MALWARE)*, Oct. 2013, pp. 116–123.

[44] G. Wichersky, T. Werner, F. Leder, and M. Schlösser, "Stormfucker: 0wning the Storm Botnet," 25th Chaos Communication Congress, Dec. 2008.

[45] J. Oikarinen and D. Reed, "Internet Relay Chat Protocol," RFC 1459, May 1993.

[46] K. Zeuge, T. Rollo, and B. Mesander, "The Client-To-Client Protocol (CTCP)," IRC Protocol Specification, Aug. 1994.

[47] J. Canavan, "The evolution of malicious IRC bots," in *Virus Bulletin Conference*, 2005, pp. 104–114.

[48] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext Transfer Protocol – HTTP/1.1," RFC 2616, Jun. 1999.

[49] J. Nazario, "BlackEnergy DDoS Bot Analysis," Arbor Networks, Tech. Rep., Oct. 2007. [Online]. Available: http://atlas-public.ec2.arbor.net/docs/BlackEnergy+DDoS+Bot+Analysis.pdf

[50] "Regin: Top-tier espionage tool enables stealthy surveillance," Symantec Corporation, Tech. Rep. Version 1.1, Aug. 2015. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/regin-analysis.pdf

[51] R. Sharpe, "Just what is SMB," V1.2, Oct. 2002.

[52] "The Duqu 2.0 Technical Details," Kaspersky lab, Tech. Rep. Version 2.1, Jun. 2015. [Online]. Available: https://securelist.com/files/2015/06/The_Mystery_of_Duqu_2_0_a_sophisticated_cyberespionage_actor_returns.pdf

[53] L. Fulton, "Eradicating the Masses & Round 1 with Phatbot?" SANS Institute, Tech. Rep., Jun. 2004. [Online]. Available: http://pen-testing.sans.org/resources/papers/gcih/eradicating-masses-1-phatbot-106262

[54] (2004, Jun.) WASTE Official Documentation & FAQ Rev. 5. [Online]. Available: http://waste.sourceforge.net/docs/docs.html

[55] J. Frankel. (2003) Waste P2P Darknet. [Online]. Available: http://waste.sourceforge.net

[56] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric," in *Peer-to-Peer Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, Mar. 2002, no. 2429, pp. 53–65.

[57] K. Kutzner and T. Fuhrmann, "Measuring Large Overlay Networks —The Overnet Example," in *Kommunikation in Verteilten Systemen (KiVS)*, ser. Informatik aktuell. Springer Berlin Heidelberg, 2005, pp. 193–204.

[58] J. Postel, "User Datagram Protocol," RFC 768, Aug. 1980.

[59] ——, "Internet Control Message Protocol," RFC 792, Sep. 1981.

[60] P. J. Criscuolo, "Distributed Denial of Service: Trin00, Tribe Flood Network, Tribe Flood Network 2000, and Stacheldraht CIAC-2319," California University Livermore Radiation Lab, Tech. Rep., Feb. 2000. [Online]. Available: http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA396999

[61] Trinoo Source Code. [Online]. Available: http://packetstormsecurity.org/distributed/trinoo.tgz

[62] F. Lau, S. H. Rubin, M. H. Smith, and L. Trajkovic, "Distributed denial of service attacks," in *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3, 2000, pp. 2275–2280 vol.3.

[63] I. Arce and E. Levy, "An analysis of the Slapper worm," *IEEE Secur. Priv.*, vol. 1, no. 1, pp. 82–87, Jan. 2003.

[64] S. Zander, G. Armitage, and P. Branch, "A survey of covert channels and countermeasures in computer network protocols," *IEEE Commun. Surv. Tutor.*, vol. 9, no. 3, pp. 44–57, Third 2007.

[65] W. Stallings, *Cryptography and Network Security: Principles and Practice*, 6th ed. Prentice Hall, 2013.

[66] B. Schneier, *Applied Cryptography: Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, 2007.

[67] "Announcing the advanced encryption standard (AES)," Standard, NIST-FIPS 197, 2001.

[68] M. Matsui, J. Nakajima, and S. Moriai, "A Description of the Camellia Encryption Algorithm," RFC 3713, Apr. 2004.

[69] R. L. Rivest, "The RC5 encryption algorithm," in *Fast Software Encryption*. Springer, Berlin, Heidelberg, Dec. 1994, pp. 86–96.

[70] D. J. Wheeler and R. M. Needham, "TEA, a tiny encryption algorithm," in *Fast Software Encryption*. Springer, Berlin, Heidelberg, Dec. 1994, pp. 363–366.

[71] R. M. Needham and D. J. Wheeler, "Tea extensions," Report, Cambridge University, Cambridge, UK, 1997.

[72] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, Aug. 2008.

[73] E. Rescorla, "HTTP Over TLS," RFC 2818, May 2000.

[74] J. Wolf, "Black Energy Crypto," Mar. 2010. [Online]. Available: https://www.fireeye.com/blog/threat-research/2010/03/black-energy-crypto.html

[75] K. Baumgartner and M. Garnaeva, "BE2 custom plugins, router abuse, and target profiles," Nov. 2014. [Online]. Available: https://securelist.com/blog/research/67353/be2-custom-plugins-router-abuse-and-target-profiles/

[76] Y. Rathore, M. K. Ahirwar, and R. Pandey, "A brief study of data compression algorithms," *Int. J. Comput. Sci. Inf. Secur.*, vol. 11, no. 10, p. 86, 2013.

[77] S. Nagaraja, A. Houmansadr, P. Piyawongwisal, V. Singh, P. Agarwal, and N. Borisov, "Stegobot: A Covert Social Network Botnet," in *Information Hiding*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, May 2011, no. 6958, pp. 299–313.

[78] A. Compagno, M. Conti, D. Lain, G. Lovisotto, and L. V. Mancini, "Boten ELISA: A novel approach for botnet C C in Online Social Networks," in *2015 IEEE Conference on Communications and Network Security (CNS)*, Sep. 2015, pp. 74–82.

[79] J. Klensin, "Simple Mail Transfer Protocol," RFC 5321, Apr. 2008.

[80] "RAT in a jar: A phishing campaign using Unrecom," Threat Advisory, Fidelis Cybersecurity Solutions, Tech. Rep., May 2014. [Online]. Available: https://www.fidelissecurity.com/sites/default/files/FTA_1013_RAT_in_a_jar.pdf

[81] "Blackenergy & Quedagh The convergence of crimeware and APT attacks," Sep. 2014, White Paper, F-Secure.

[82] R. Samani and C. Beek, "Updated BlackEnergy Trojan Grows More Powerful," Jan. 2016. [Online]. Available: https://blogs.mcafee.com/mcafee-labs/updated-blackenergy-trojan-grows-more-powerful/

[83] G. Lawton, "On the Trail of the Conficker Worm," *Computer*, vol. 42, no. 6, pp. 19–22, Jun. 2009.

[84] P. Porras, H. Saidi, and V. Yegneswaran, "An analysis of conficker's logic and rendezvous points," Technical report, SRI International, Tech. Rep., 2009.

[85] GReAT. (2015, Jun.) The Duqu 2.0 persistence module. Securelist. [Online]. Available: https://securelist.com/blog/research/70641/the-duqu-2-0-persistence-module/

[86] D. Dittrich and S. Dietrich, "Command and control structures in malware," *Usenix Mag.*, vol. 32, no. 6, 2007.

[87] M. Kaczmarek, "Malware Instrumentation Application to Regin Analysis," *J. Cybercriminalité Investig. Numér.*, vol. 1, no. 1, pp. 1–12, Dec. 2015.

[88] "The Regin Platform Nation-State Ownage of GSM Networks," Kaspersky lab, Tech. Rep. Version 1.0, Nov. 2014. [Online]. Available: https://cdn.securelist.com/files/2014/11/Kaspersky_Lab_whitepaper_Regin_platform_eng.pdf

[89] G. Bonfante, J. Y. Marion, and F. Sabatier, "Gorille sniffs code similarities, the case study of qwerty versus regin," in *2015 10th International Conference on Malicious and Unwanted Software (MALWARE)*, Oct. 2015, pp. 82–89.

[90] K. Chiang and L. Lloyd, "A case study of the rustock rootkit and spam bot," in *The First Workshop in Understanding Botnets*, vol. 20, 2007.

[91] David Nselmi, Richard Boscovich, T.J. Campana, Samantha Doerr, Marc Lauricella, Oleg Petrovsky, Tareq Saade, and Holly Stewart, "Battling the Rustock Threat," Microsoft, Tech. Rep., 2011. [Online]. Available: https://lammgl.files.wordpress.com/2011/03/battling-the-rustock-threat_english.pdf

[92] "WORM:LINUX/SLAPPER Threat description," 2002. [Online]. Available: https://www.f-secure.com/v-descs/worm_linux_slapper.shtml

[93] P. Porras, H. Saidi, and V. Yegneswaran, "A Multi-Perspective Analysis of the Storm (Peacomm) Worm," Computer Science Laboratory, SRI International, Tech. Rep., Oct. 2007. [Online]. Available: http://www.cyber-ta.org/pubs/StormWorm/report/

[94] D. Jackson. (2007, Sep.) Analysis of Storm Worm DDoS Traffic. SecureWorks. [Online]. Available: http://web.archive.org/web/20101129235652/http://secureworks.com/research/blog/index.php/2007/09/12/analysis-of-storm-worm-ddos-traffic/

[95] Mixter. Tribe Flood Network Source Code. [Online]. Available: http://web.archive.org/web/20010215035009/http://packetstorm.securify.com/distributed/tfn.tgz

[96] J. Baltazar, J. Costoya, and R. Flores, "Infiltrating WALEDAC Botnet's Covert Operations," Trendmicro, Tech. Rep., 2013. [Online]. Available: http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp_infiltrating_the_waledac_botnet_v2.pdf

[97] K. McNamee, "Malware Analysis Report Botnet: ZeroAccess/Sirefef," Kindsight Security Labs, Tech. Rep., Feb. 2012. [Online]. Available: http://web.archive.org/web/20140710070427/http://www.kindsight.net/sites/default/files/Kindsight_Malware_Analysis-ZeroAcess-Botnet-final.pdf

[98] S. Hittel and R. Zhou, "Trojan.ZeroAccess Infection Analysis," Symantec Corporation, Tech. Rep., 2012. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/trojan_zeroaccess_infection_analysis.pdf

[99] N. Falliere and E. Chien, "Zeus: King of the Bots," Symantec Corporation, Tech. Rep., 2009. [Online]. Available: https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/zeus_king_of_bots.pdf

[100] Zeus Source Code. [Online]. Available: https://github.com/Visgean/Zeus/

[101] R. Radvanovsky and J. Brodsky, *Handbook of Scada/Control Systems Security*. Crc Pr Inc, Jan. 2013.

[102] O. Thonnard, L. Bilge, G. O'Gorman, S. Kiernan, and M. Lee, "Industrial Espionage and Targeted Attacks: Understanding the Characteristics of an Escalating Threat," in *Research in Attacks, Intrusions, and Defenses*. Springer, Berlin, Heidelberg, Sep. 2012, pp. 64–85.

[103] J. Postel and J. Reynolds, "FILE TRANSFER PROTOCOL (FTP)," RFC 959, Oct. 1985.

[104] Z. Micskei and H. Waeselynck, "The many meanings of UML 2 Sequence Diagrams: A survey," *Softw Syst Model*, vol. 10, no. 4, pp. 489–514, Oct. 2011.

[105] Object Management Group, "Unified Modeling Language," Mar. 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5

[106] H. Berghel, "A Farewell to Air Gaps, Part 1," *Computer*, vol. 48, no. 6, pp. 64–68, Jun. 2015.

[107] M. Leech, M. Ganis, Y. Lee, R. Kuris, D. Koblas, and L. Jones, "SOCKS Protocol Version 5," RFC 1928, Mar. 1996.

Gernot Vormayr received his B.Sc. degree in electrical engineering and his Dipl.-Ing. degree (M.Sc. equivalent) in computer technology from the TU Wien, Vienna, Austria. Since 2016, he has been employed as a University Assistant at the Institute of Telecommunications, TU Wien, where he is pursuing his doctoral degree. His current research interest is in malware communication with a focus on botnets.

Tanja Zseby received her diploma degree (Dipl.-Ing.) in electrical engineering and her doctoral degree (Dr.-Ing.) from TU Berlin, Germany. She worked as Scientist and Head of the Network Research Group at the Fraunhofer Institute for Open Communication Systems, Berlin, Germany and as a visiting scientist at the University of California, San Diego (UCSD). She now is a full professor of communication networks at TU Wien.

Joachim Fabini received his Dipl.-Ing. degree in computer sciences and his Dr.Techn. degree in electrical engineering from the TU Wien. After five years of research and development with Ericsson Austria, he joined the Institute of Telecommunications, TU Wien, in 2003. He is a Senior Scientist with the Communication Networks Group with research focus on active measurement methodologies.