A Multifaceted Approach to Understanding the Botnet Phenomenon

Moheeb Abu Rajab Jay Zarfoss Fabian Monrose Andreas Terzis
Department of Computer Science, Johns Hopkins University
Baltimore, Maryland, USA
moheeb@cs.jhu.edu, zarfide@gmail.com, fabian@cs.jhu.edu, terzis@cs.jhu.edu

ABSTRACT

The academic community has long acknowledged the existence of malicious botnets, however to date, very little is known about the behavior of these distributed computing platforms. To the best of our knowledge, botnet behavior has never been methodically studied, botnet prevalence on the Internet is mostly a mystery, and the botnet life cycle has yet to be modeled. Uncertainty abounds. In this paper, we attempt to clear the fog surrounding botnets by constructing a multifaceted and distributed measurement infrastructure. Throughout a period of more than three months, we used this infrastructure to track 192 unique IRC botnets of size ranging from a few hundred to several thousand infected end-hosts. Our results show that botnets represent a major contributor to unwanted Internet traffic—27% of all malicious connection attempts observed from our distributed darknet can be directly attributed to botnet-related spreading activity. Furthermore, we discovered evidence of botnet infections in 11% of the 800,000 DNS domains we examined, indicating a high diversity among botnet victims. Taken as a whole, these results not only highlight the prominence of botnets, but also provide deep insights that may facilitate further research to curtail this phenomenon.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection—*Invasive Software*

General Terms

Security, Measurement

Keywords

Botnets, Computer Security, Malware, Network Security

1. INTRODUCTION

Despite the fact that botnets first appeared several years ago, they have only recently sparked the interest of the research community. The term *botnets* is used to define networks of infected end-hosts, called *bots*, that are under the control of a human operator commonly known as a *botmaster*. While botnets recruit vulnerable machines using methods also utilized by other classes of malware (*e.g.,* remotely exploiting software vulnerabilities, social engineering, etc.), their defining characteristic is the use of *command and control* (C&C) channels. The primary purpose of these channels is to disseminate the botmasters' commands to their bot armies. These channels can operate over a variety of (logical) network topologies and use different communication mechanisms, from established Internet protocols to more recent P2P protocols. However, the vast majority of botnets today use the Internet Relay Chat (IRC) protocol [13] which was originally designed to form large social chat rooms.

While other classes of malware were mostly used to demonstrate technical prominence among hackers, botnets are predominantly used for illegal activities. These activities range from extortion of Internet businesses to e-mail spamming, identity theft, and software piracy. Unfortunately, even with the substantial increase in botnet activity witnessed over the past few years, little is known about the specifics of this malicious behavior. For instance, questions pertaining to the prevalence of botnet activity, the number of different botnet subspecies (and how they can be behaviorally categorized), and the evolution of a botnet over its lifetime, abound.

This paper presents the results of our effort to address these questions. We argue that a thorough and complete understanding of this problem calls for a multifaceted measurement approach. Furthermore, we believe that this approach must capture the behavior and impact of botnets from multiple viewpoints. In that regard, this paper makes two key contributions, namely **(1)** the development of a multifaceted infrastructure to capture and concurrently track multiple botnets in the wild, and **(2)** a comprehensive analysis of measurements reflecting several important structural and behavioral aspects of botnets. The infrastructure we developed synthesizes multiple data collection or "sensing" techniques, including distributed malware collection points to capture botnet binaries, IRC tracking to gain an insider perspective of the behavior of live botnets, and DNS cache probing to assess the global prevalence of botnets. We were able to observe more than two hundred botnets and actively track more than a hundred long-lived ones over a period of more than three months. By cross checking the multiple views gained by the different sensing techniques we reveal a number of behavioral and structural features of botnets not previously reported in the literature.

The remainder of the paper is organized as follows: Section 2 provides background information on botnets and highlights the challenges associated with botnet detection and tracking. We present the approach we developed to infiltrate large numbers of botnets in Section 3, and describe a novel approach for extracting infor-

IMC'06, October 25–27, 2006, Rio de Janeiro, Brazil.

mation from the malicious binaries collected using our distributed infrastructure. Section 4 presents the analysis of the collected data. Related work is presented in Section 5. We conclude in Section 6.

2. BACKGROUND

A botnet is a group of infected end-hosts under the command of a botmaster. Figure 1 illustrates the various stages in a typical botnet life-cycle. Botnets usually commandeer new victims by remotely exploiting a vulnerability of the software running on the victim. Botnets borrow infection strategies from several classes of malware, including self-replicating worms, e-mail viruses, etc. Infections can also be spread by convincing victims to run some form of malicious code on their machines (*e.g.,* by executing an email attachment).

Once infected, the victim typically executes a script (known as *shellcode*) that fetches the image of the actual bot binary from a specified location.[1] Upon completion of the download, the bot binary installs itself to the target machine so that it starts automatically each time the victim is rebooted.

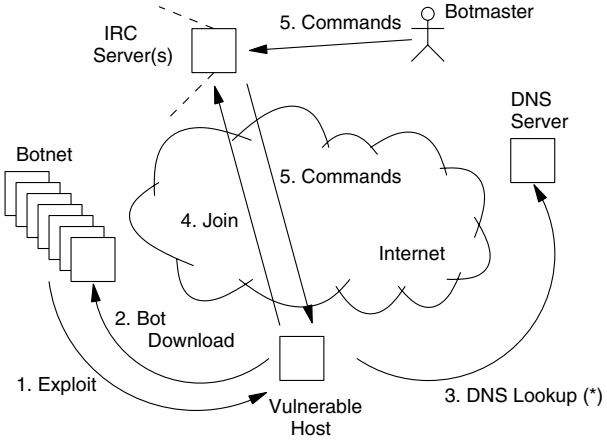


Figure 1: The life-cycle of a typical botnet infection. Steps with enclosed asterisk are optional.

As mentioned earlier, the defining characteristic of botnets is manifested by the fact that individual bots are controlled via commands sent by the network's botmaster. The communication channel used to issue commands can be implemented using a variety of protocols (*e.g.*, HTTP, P2P, etc.). However, the majority of botnets today use the Internet Relay Chat (IRC) protocol [13]. The IRC protocol was specifically designed to allow for several forms of communication (point-to-point, point to multi-point, etc.) and data dissemination among large number of end-hosts. The inherent flexibility of this protocol, as well as the availability of several open-source implementations, enables third parties to extend it in ways that suit their needs. These features make IRC the protocol of choice for botmasters, as it simplifies the botnet implementation and provides a high degree of control over the bots. Therefore, for the rest of this paper, we focus our attention to IRC-based botnets since IRC is the prominent botnet control mechanism in use today.

Upon initialization, each bot attempts to contact the IRC server address given in the executable. In many cases, this step requires resolving the DNS name of the IRC server (Step 3 in Figure 1). Using a DNS name instead of a hard-coded IP address allows the botmaster to retain control of her botnet in the event that the current IP address associated with the DNS name of the IRC server gets black-listed.

Once the IP address of the IRC server is available, the bot attempts to establish an IRC session with the server and joins the command and control channel specified in the bot binary. Generally speaking, the bot-to-IRC server communication requires any combination of three types of authentication. First, a bot needs to authenticate itself to the IRC server using the `PASS` message in order to successfully begin the IRC session with that server. Second, botmasters normally protect the command and control channel with a password and hence require the bot to authenticate itself before joining. The passwords corresponding to these two authentication phases are contained in the bot binary and authentication normally takes place in the clear. The third type of authentication, which is not part of the IRC protocol, requires the botmaster to authenticate herself to the bot population before she is able to issue any command. While the first two authentication steps are intended to thwart outsiders from joining the C&C channel, the last authentication phase aims to protect bots from being overtaken by other botmasters attempting to seize control of fledgling botnets.

Once the bot successfully joins the specified IRC channel, it automatically parses and executes the channel topic. The topic contains the default command that every bot should execute. Depending on the channel mode set by the botmaster, bots *might* be able to "hear" all messages exchanged on the channel. This broadcast behavior of IRC channels is a design feature that makes the IRC protocol suitable for supporting large-scale chat rooms. As we show later, broadcast via the C&C channel is an invaluable source of insider information about the activities and capabilities of some botnets. While convenient, one cannot rely on this feature for information extraction, since in some cases, IRC servers disable it either to reduce "chatter" or to limit communication overhead.

The steps described so far are shared among all IRC-based botnets. The degree of commonality however ends here; different botnets express the set of the commands and responses exchanged between the botmaster and her bots as extensions on top of the standard IRC protocol. While the syntax of these commands follow the same general structure, they do vary across different botnets. This variability is primarily a result of botmasters' desire to "personalize" their bots, and in doing so, complicate the task of tracking these botnets in an intelligent manner. Moreover, the repertoire of available commands elicit a wide variety of responses, which in turn, greatly complicates the classification of botnet behaviors.

3. MEASUREMENT METHODOLOGY

As Figure 2 illustrates, our data collection methodology encompasses three logically distinct phases: **(1)** malware collection, **(2)** binary analysis via gray-box testing and, **(3)** longitudinal tracking of IRC botnets through IRC and DNS trackers.

The goal of the malware collection phase is simply to collect as many bot binaries as possible. However, developing a scalable and robust infrastructure to achieve this goal is a challenging problem in its own right, and has been the subject of numerous research initiatives (*e.g.*, [20, 23]). In particular, any malware collection infrastructure must support a wide array of data collection endpoints and should be highly scalable. Additionally, special measures must be implemented to prevent any part of the system from participating in malfeasance. Our system design and implementation draw on experience learned from earlier work [1, 2, 10, 19, 20, 23], but include several additions that are unique to our goals. In what follows, we discuss the specifics of our infrastructure.

[1] We observed that botnets using random scanning to spread, usually serve the bot binary from the same machine that exploited the remote vulnerability in the first place.

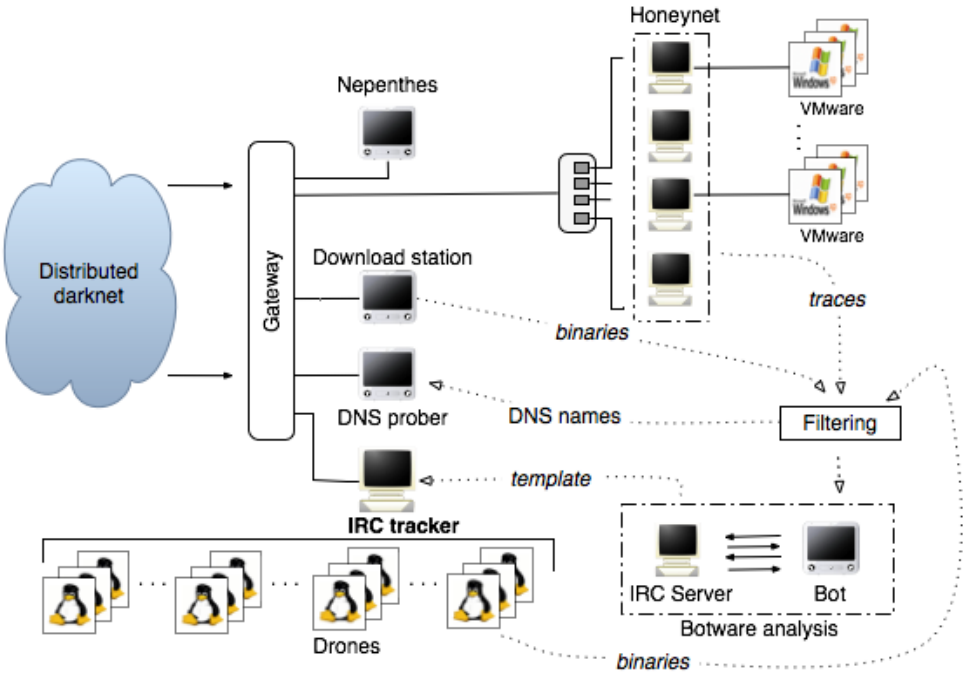


Figure 2: Overall data collection architecture.

MALWARE COLLECTION. As we show later, a significant portion of botnet-related spreading activity is localized, targeting certain parts of the IP space. Any single vantage point is thus likely to miss substantial portions of such scanning activity. We attempt to minimize this undesired effect by deploying our collection architecture on a conglomeration of distributed darknets [2]. This collection includes a large locally deployed darknet and 14 distributed nodes using the PlanetLab testbed [18]. These nodes have access to darknet IP space located in ten different /8 prefixes.

In this distributed darknet, we deploy a modified version of the `nepenthes` platform [2]. In short, `nepenthes` mimics the replies generated by vulnerable services in order to collect the first stage exploit (typically a Windows shellcode). In the case of the PlanetLab nodes, several modifications to `nepenthes` were necessary. For one, these nodes are setup to deliver traffic destined to the darknet as raw packets through a special *proxy* interface. However, since `nepenthes` does not support raw sockets, packet translation is required to transform the raw packets and inject them to a local tunneling interface. To do so, we configured `nepenthes` to bind to the tunneling interface using regular sockets and receive the packets via a translation module written in `Click` [15]. Moreover, since PlanetLab nodes do not allow user-level processes to bind to privileged ports, the `Click` module also performs port translation. The process is shown pictorially in Figure 3.

To prevent excessive downloads and reflection attacks caused by "heavy hitters" requesting the same URL multiple times, we disable the on-line download modules in `nepenthes`. Instead, we generate a list of the URL targets to be downloaded, and send this list to a machine designated this task. This download station, filters the list and extracts the unique sources and URLs. All previously unseen URL targets are subsequently downloaded.

[2]The term darknet is used to denote an allocated but unused portion of the IP address space.

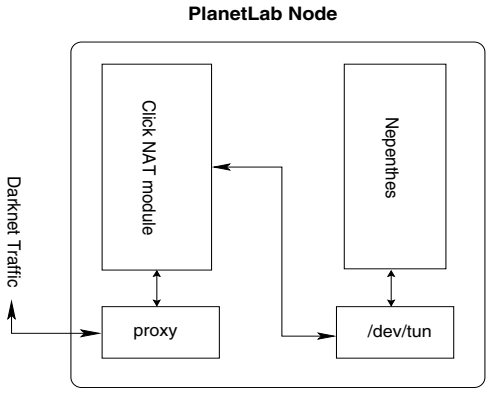


Figure 3: PlanetLab configuration.

Additionally, to complement the role of `nepenthes`, we make use of a honeynet. The primary reason for doing so is to ensure catching exploits missed by `nepenthes`. These failures are most likely due to the responder's inability to mimic unknown exploit sequences or to parse certain shellcodes. Currently, our honeynet is composed of a number of honeypots running unpatched instances of Windows XP in a virtualized environment [21]. Each honeypot instance is assigned a static private-space IP address on a separate VLAN. Infected honeypots are allowed to sustain IRC connections with unique botnet IRC servers until the virtual machines are re-imaged. At that point, all suspect binaries are retrieved by comparing the disk contents of the virtual machine to a clean Windows image. As with the binaries collected by the responders, the binaries retrieved from the honeynet are also sent to an analysis engine for graybox testing (discussed in Section 3.1).

GATEWAY. The gateway supports a myriad of functions, key among those being to route darknet traffic to various parts of the internal network. In our current setting, the gateway forwards traffic destined to eight /24 prefixes from the local darknet. The set of selected prefixes is automatically rotated every day to ensure full coverage of the darknet. At present, half of the monitored prefixes are directed to the local responder, and the other half to the honeynet. In order to keep the number of required honeypots small, we use Network Address Translation (NAT) to map each honeypot to 128 external darknet IP addresses. The use of NAT also reduces management overhead as the IP addresses of the honeypots need not be changed each time the address space is rotated.

The gateway also serves as a firewall whose main task is to prevent the honeypots from engaging in any outbound attacks or infecting each other. Cross-infections among honeypots is prevented by configuring each honeypot on a separate Virtual LAN (VLAN) and terminating any traffic across VLANs at the gateway. The firewall prevents abuse by rate-limiting any allowed outbound connections and blocking all outbound traffic on popular vulnerable ports (*e.g.*, 135, 139, 445). All remaining outbound traffic is queued to a detection process that is configured to allow infected honeypots to follow the typical infection sequence outlined in Section 2.

Specifically, we exploit the fact that bots continuously attempt to contact their IRC servers and run an IRC detection module at the gateway. This module is implemented as an extension to snort_inline [16], and its main function is to detect and manage the IRC connections. In short, we issue a `SYN-ACK` for the first connection attempts from the honeynet; once a connection is established, the detection module searches the application-level traffic for common IRC protocol strings used during the server handshake (*e.g.*, `NICK, JOIN, USER`). After a valid IRC connection attempt is witnessed, the detection module establishes a record for that IRC session and sends a `RST` to the originating honeypot. When the honeypot subsequently attempts to reconnect, the connection is allowed to proceed to the actual IRC server—subject of course, to rate limiting. Note that since *all* outbound connections are analyzed at the application level, we can detect IRC traffic on non-standard IRC ports as well.

Several additional measures were implemented to maximize the information gain from our infrastructure. For instance, the detection module only allows one honeypot to connect to a particular IRC server at any point in time. Since our honeypots run on resource-limited virtual machines, having multiple malware instances running on a honeypot is destabilizing. To avoid this problem, the gateway detects when a honeypot has been infected, and dynamically inserts rules to block further inbound attack traffic towards that honeypot.

Lastly, the gateway performs several miscellaneous control and management tasks for different components of our architecture. These tasks include triggering periodic re-imaging of the honeypots, loading clean Windows images, pre-filtering and control functionality for the download station, and running a local DNS server to resolve DNS queries from the honeypots.

3.1 Binary analysis via graybox testing

We use graybox analysis to extract the features of suspicious binaries (regardless of the mechanism by which they were collected). The analysis spans two logically distinct phases performed on an isolated network segment. The first phase is aimed at deriving a network fingerprint of the binary under scrutiny, while the second attempts to extract its IRC-specific features. Since none of these actions use debuggers, disassemblers, or other interactive tools traditionally used in binary analysis, both phases are easily automated. To aid the analysis, we isolate part of our private network to contain a server (configured as a sink for all network traffic) and a virtualized client machine. Each collected binary is executed on a clean image of Windows XP instantiated as a virtual machine on the client. All network traffic is logged during the inspection period, and the following actions are performed:

Phase 1: Creation of a network fingerprint. Since the server acts as a network sink, any network activity initiated by the suspected malware will be detected. Moreover, as each new instance runs within a clean image of Windows XP, any benign network activity can be easily filtered. The traffic logs are automatically processed to extract a network fingerprint, $f_{net} = \langle \texttt{DNS}, \texttt{IPs}, \texttt{Ports}, \texttt{scan} \rangle$, representing the targets of any DNS requests, the destination IP addresses, the contacted ports (and protocols), and whether or not default scanning behavior was detected, respectively. We define default scanning behavior as any attempt to contact more than a predetermined threshold of n (=20) distinct destinations on the same port during the monitored period. At present, this stage takes approximately six minutes to complete per binary, including the time to reload the virtual machine.

Phase 2: Extraction of IRC-related features. While the previous phase extracts the network-level characteristic of an unknown binary, the goal of the second phase is to identify the binary's application level behavior. To extract IRC-related features we instantiate a modified version of the UnrealIRC daemon [22] on the network sink. This IRC server listens on all ports ever observed in the network fingerprint created during Phase 1. This is important as it is common to observe IRC connections on non-standard ports. As before, a clean image of Windows XP is loaded before inspecting the binary at hand. When an IRC connection is detected, our modified server creates an IRC-fingerprint, $f_{irc} = \langle \texttt{PASS}, \texttt{NICK}, \texttt{USER}, \texttt{MODE}, \texttt{JOIN} \rangle$, representing an initial password to establish an IRC session with the server, the format of the nickname and username chosen by the bot, the particular modes set, and which IRC channels are automatically joined (with associated channel passwords). Taken together, f_{net} and f_{irc} provide enough information to join a botnet in the wild. However, in order to mimic an actual bot behavior we need to learn the botnet's "dialect" (*i.e.*, the syntax of the botmaster's commands as well as the corresponding responses sent by the actual bot).

In order to learn a botnet "dialect" we make the bot connect to our local IRC server. Once connected, the bot is forced into a default channel. Next, an IRC query engine is dynamically loaded. From that point, our query engine essentially plays the role of a botmaster. That is, for a given bot, we learn how to correctly mimic its behavior in the wild by subjecting it to a barrage of commands. This set of commands includes all the IRC commands that we originally observed in our honeynet traces as well as the commands extracted from the publicly available source code of known bots analogous to the analysis in [3].

The observant reader may wonder how the query engine can coerce the bot into communicating since the query engine may not be able to authenticate itself to the bot. As we noted earlier, botmasters must generally authenticate themselves using a unique password *before* the bot will be responsive [19]. This authentication information can be automatically extracted from the log *if* the bot was observed on a honeypot. Fortunately, in the cases where no log exists, we can coerce these bots into communicating with us via a very simple tactic: the standard behavior of IRC-based bots is to parse the server's channel topic message (`RPL_TOPIC`) [13] and execute its instructions, with no authentication. This command is

normally only sent when a bot first joins a channel, and running instructions from this command allows the botmaster's minions to become productive as soon as they connect to the C&C server. Hence, we modified our IRC server to allow the query engine to send `RPL_TOPIC` notifications on the fly. Bots accordingly parse and execute the commands, without the need for authentication or any requirement to restart the virtual machine instance.

The output of the querying process is a command-response *template* that captures the "dialect" of the bot. This template is later fed to our IRC tracker (discussed shortly). This learning component is a core part of our architecture, and allows for a stealthy, longitudinal study of botnet dynamics.

3.2 Longitudinal Tracking of Botnets

As Figure 2 indicates, we track botnets in our study by two independent means: an "insider's perspective" made possible by a custom lightweight IRC tracker and by probing DNS caches across the globe. The underpinnings of each method are discussed below.

3.2.1 IRC Tracker: A Look From Within

The IRC tracker (also called a *drone*) is a modified IRC client that can join a specified IRC channel and automatically answer directed queries based on the template created by the graybox testing technique. Specifically, given the fingerprint f_{irc} and a template, the IRC tracker instantiates a new IRC session to the actual IRC server. At this point, the drone operates in the wild, and pretends to dutifully follow any commands from the botmaster, and provides realistic responses to her commands. Clearly, our IRC trackers need to be intelligent enough that they appear as responsive and powerful bots—otherwise, our drones may be exempted from partaking in interesting behaviors. In order to appear as real as possible, several non-trivial tasks must first be accomplished during the template generation phase.

First, traffic must be "filtered" so that inappropriate information is not included in the template. This filtering is performed automatically while the actual bot software is executing. For example, computer statistics such as memory and disk space are changed to resemble values consistent with the hardware and software specifications of a real machine. Second, and more importantly, nearly all bot software is statefull. Hence, a command, for example, to stop a scan will usually result in a different reply depending on whether or not scanning was already running. To address this, the IRC query engine exposes the different responses by issuing sets of commands that require statefull responses in varying combinations. The IRC tracker is designed to mimic these state changes when it is in the wild so that it responds appropriately.

The efficiency of our approach allows us to run a large number of drones on a single machine. To improve our mimicry, the IRC tracker joins and leaves the tracked channels at random intervals.[3] Once a drone's staying interval expires, that drone leaves the server for a random interval (of no more than 10 minutes) after which it restarts and rejoins the same channel under a different user ID that follows the naming convention inscribed in the template. In addition, due to the address translation occurring at the gateway, each newly instantiated drone is assigned a different external IP address.

3.2.2 DNS Tracking

We gain a second perspective by exploiting the fact that most bots issue DNS queries to resolve the IP addresses of their IRC servers. Specifically, we probe the caches of a large number of DNS servers in order to infer the footprint of a particular botnet, defined here as the total number of DNS servers giving cache hits. A cache hit implies that at least one client machine has queried the DNS server within the lifetime (`TTL`) of its DNS entry. While DNS cache probing has been used recently for a number of purposes [7, 9, 14], we are not aware of any prior effort that used cache probing to infer a botnet's footprint.

Our original list of 1.6 million DNS servers, denoted $\mathcal{D}$, was obtained by collecting the `NS` records of the DNS domains extracted from a large list of crawled URLs [17]. The list was then subject to a number of sanitization steps. First, we filter all name servers for sensitive Top Level Domains (TLDs) (*e.g.*, `.gov`, `.mil`). Next, we apply additional filtering to check the consistency of each name server's replies. Namely, for each server in $\mathcal{D}$, we send two consecutive DNS queries for an existing known DNS name and inspect the replies. The first is a recursive query that forces the DNS server to resolve the query completely. The second query is sent with the recursion flag turned off to elicit a local answer from the server's cache. We compare the replies for consistency and also validate that the value of the `TTL` field in the second response is smaller than the one in the first response. All the servers that fail any of these checks, as well as those that did not respond to our queries, are removed from the list. At the end of this process we are left with $\sim$ 800,000 name servers[4], denoted $\hat{\mathcal{D}}$, which we use as our master list.

The DNS probing experiments are then carried out from a number of machines assigned this task. Each DNS name of a newly detected IRC server is added to the list of servers to be probed, denoted here as $\mathcal{T}$. For a given IRC server $t \in \mathcal{T}$, we probe the caches of all DNS servers in $\hat{\mathcal{D}}$ and record any cache hits. We deliberately set a low querying rate so that no DNS server is unduly burdened. The average probing rate for a single DNS server is about 20 queries per day. More recently detected IRC server names are given priority and are queried more frequently.

Clearly, the number of cache hits for any entry in $\mathcal{T}$ is a lower bound on its true DNS footprint. The discrepancy is due to the fact that we will only be able to record a cache hit if a bot made a lookup query to its local DNS server, and that entry was cached at the time of our probe. Additionally, a cache hit only indicates that at least one bot issued a DNS query for that IRC server, but does not reveal how many bots actually queried the name server within the TTL. Finally, even though the list of DNS servers we query is large, it is only a subset of the total number of nameservers on the Internet. With these caveats in mind, we consider the DNS probing results as a mere indication of the relative footprints created by different botnets. As we show later, although the results derived from DNS probing are course grain, DNS probing provides an important secondary avenue for tracking botnets that disable the broadcast feature on their IRC channels.

4. RESULTS AND ANALYSIS

In what follows, we present the results and lessons learned by integrating information from the various data collection channels presented in the previous section. We report on data that we started collecting on 2/1/2006 and include **(1)** traffic traces captured at our local darknet over a period of more than three months, **(2)** IRC logs gathered over the span of three months—covering data from more than 100 botnet channels either visited by our IRC tracker or observed on our honeynet, and **(3)** results of DNS cache hits from tracking 65 IRC servers for more than 45 days. Unlike previ-

[3]Currently set to 2 hours with a randomized maximum drift of 25%.

[4]A handful of servers were also removed after requests from their respective network operators to not participate in the study.

ous botnet-related studies (*e.g.*, [5, 6]), we focus on correlating the results from these viewpoints in order to gain a deeper understanding of botnet activity in general. Therefore, whenever applicable, we cross-reference measurements acquired from different measurement techniques to provide a more complete picture of this phenomenon.

4.1 Prevalence of the Botnet Phenomenon

In order to provide a high level view of botnet prevalence, we highlight the contribution of their spreading activity to the overall unwanted traffic received at our darknet space. In particular, we examine to what extent the traffic received at this viewpoint can be attributed to botnets spreading activity. Additionally, we provide cumulative statistics from our DNS probes showing the total number of potentially infected networks as well as how these networks are distributed among the top-level domains.

4.1.1 Botnet Traffic Share

Figure 4 shows a two week snapshot of the total incoming `SYN` packets to our local darknet versus those originating from known botnet spreaders. Simply speaking, a botnet spreader is any source that successfully completed an exploit transaction that delivered a bot executable. We extract these botnet spreaders from our traffic traces by mapping all collected bot binaries to the sources that deliver them. This traffic most likely represents botnets scanning to find new victims.

A number of immediate observations can be inferred from Figure 4. For one, a rudimentary count of the total number of `SYN` packets shows that $\approx 27\%$ of the incoming `SYN`s can be attributed to known botnet spreaders. If we consider the traffic directed to target ports commonly used by botnet spreaders (*e.g.*, 135, 139, 445, 3127), then the total share increases to 76%. Of course, it is well known that a wide variety of non-botnet related malware, such as worms, may also target these ports, so the precise botnet traffic share can not be easily determined. However, it is interesting that many of the peaks in the total traffic are synchronized with peaks in the botnet-related traffic.

A closer look at one of these peak periods, shown in Figure 5, clearly shows a distinct alignment between traffic corresponding to known botnet spreaders and all the traffic received at the darknet. Upon further examination of the darknet traces, we found that more than 90% of all the traffic during that peak targeted ports used by botnet spreaders. Additionally, for the sources that succeeded in sending exploit shellcode, we also examined the active responder logs for similarities among the downloaded exploits. Interestingly, more than 70% of the sources during peak periods sent shell exploits similar to those sent by the botnet spreaders. Similar patterns were also observed in the traffic traces collected from the PlanetLab nodes. While these results do not provide an accurate estimate for the *overall* botnet traffic, the amount of botnet-related traffic is certainly greater than 27%.

4.1.2 Botnet Prevalence: A Global look

To provide a broader view of the scope of botnet activity, we present the cumulative results of our DNS probing experiments. Over the duration of the monitored period, we tracked cache hits for a total of 65 IRC server domain names. From the 800,000 probed servers, 85,000 (or $\approx 11\%$) were involved in at least one botnet activity.[5]

Table 1 presents a breakdown of the top level domains for which botnet activity was detected. For example, 55% of the servers in

[5]We consider hits from the primary and secondary nameservers of the same domain as a single hit.

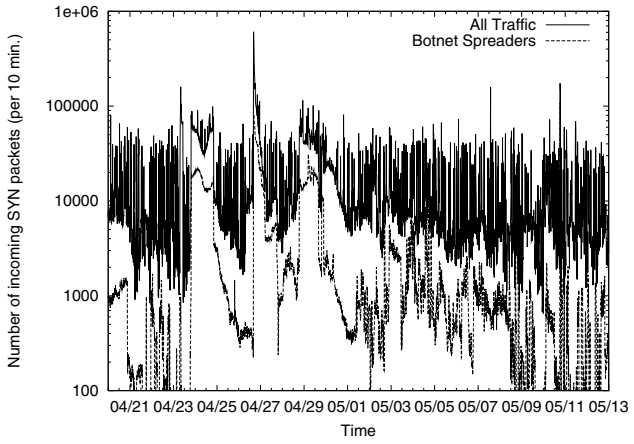


Figure 4: Time series of incoming `SYN` packets to the darknet.

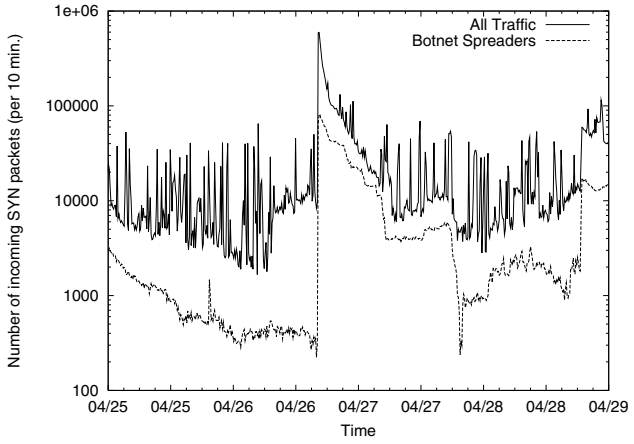


Figure 5: Zoomed view showing synchronized peaks.

our dataset are for `.com` domains, and 82% of the overall DNS cache hits were from name servers in that TLD. Moreover, 29% of the `.com` servers (in our data) had at least 1 cache hit. Interestingly, although the total fraction of servers from the `.cn` TLD is small (0.2%), 95% of these servers (that is, greater than 1500) showed evidence of botnet activity for the clients they serve. Figure 6 depicts examples of the widespread presence of botnet activity on the Internet.[6] The mapping of IP address to geographic location is based on the IP2Location dataset [11], while the derived locations are displayed using the GoogleMaps™API. In the section that follows we illustrate the size and the evolution of DNS footprints for individual botnets.

4.2 Botnets Spreading and Growth patterns

As noted earlier, botnets use a variety of means to spread and recruit new victims, including (but not limited to) email, web and active scanning to exploit vulnerable services. Scanning is by far the most prevalent spreading mechanism. Unlike traditional worms that exhibit a monotonic scanning behavior, botnets exhibit behavior that varies across different classes of botnets as well as temporally for a single botnet. For the most part, botnets observed in the

[6]Additional maps can be found at `http://hinrg.cs.jhu.edu/botnets/`.

TLD	Fraction of svrs probed	Percentage of all cache hits	Normalized hit ratio
.com	.55	82%	29%
.net	.134	5.5%	8.1%
.kr	.015	3.2%	40%
.org	.037	2.4%	13%
.cn	.002	0.9%	95%
.ru	.017	0.6%	7.3%
.de	.016	0.48%	6%
.edu	.01	0.4%	8%
.ro	.004	0.32%	0.4%
.jp	.022	0.25%	2.2%
other	.21	4.45%	N/A

Table 1: TLD statistics of DNS servers supporting clients involved in at least one botnet.

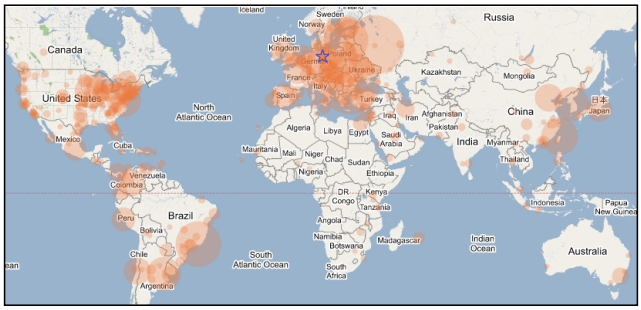


Figure 6: Geographic location of the DNS cache hits for one of the tracked botnets. The star indicates the location of the IRC server.

wild can be grouped into two broad types, (**I**) worm-like botnets that continuously scan certain ports following a specific target selection algorithm and (**II**) botnets with variable scanning behavior. In the second case, bots are equipped with a number of scanning algorithms (*e.g.*, uniform, non-uniform, localized) and only scan after receiving a command over the command and control channel.

Of the 192 IRC bots we captured, 34 were Type-I (*i.e.*, exhibited a worm-like behavior). Upon infection, a bot of this type immediately starts scanning the IP space looking for new victims. Additionally, these bots initiate connections to a hard-coded list of DNS names corresponding to IRC servers (some of which may be public). We found that all these IRC servers and/or the channels the bots tried to join were unreachable—either because the channel was banned by the public IRC server, or because the DNS name did not resolve to a valid IP address. However, due to their unrelenting scanning activity, the infected population of such worm-like botnets continue to grow over time, and may become fairly large in size. Indeed, Dagon *et al.* [6] reported botnets exhibiting this same behavior with a footprint of up to 350,000 infected machines. Finally, because the IRC servers corresponding to Type-I botnets were for the most part unavailable, they were exempted from our DNS probes.

Type-II botnets are the more prevalent class seen today. As such, the majority of our analysis is focused on this class. Such botnets are much more difficult to track due to their intermittent and continuously changing behavior. This highly fluid nature also complicates the task of generating signatures that can identify their activity with high accuracy.

Table 2 summarizes the most common scanning practices used by Type-II botnets. As the table shows, localized and non-uniform scanning are the predominant scanning techniques. When botmasters become active, 28% of their commands are scan related. By contrast, 80% of the time, the default channel topic is to scan. Approximately 85% of the botmaster-issued scan commands are targeted to a specific prefix /8 or /16 prefix. Additionally, notice that most of the targeted scanning activity is aimed at /8 network prefixes, while the localized scanning mostly targets the local /16 subnet of the victim. Finally, we note that bots are equipped with a set of flexible options to fine tune their scanning activity. These options include choosing the target vulnerability, the scanning rate, the number of threads to use, the number of packets to send, and the duration of scanning. This flexibility produces heterogeneous botnet growth patterns that are distinct from those created by worms or Type-I botnets.

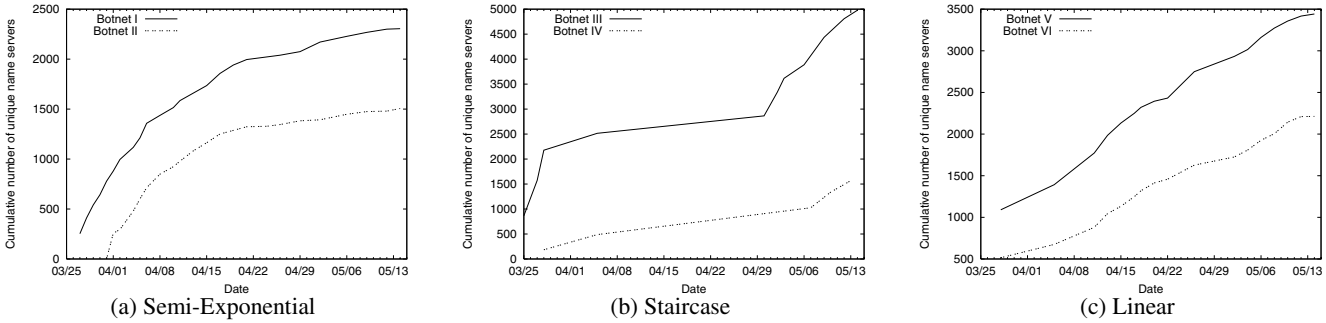
	Default Topic	Botmaster Command
Localized scanning	66%	15%
- `Class A`	11%	18%
- `Class B`	89%	82%
Targeted scanning	32%	87.4%
- `Class A`	80%	88%
- `Class B`	20%	12%
Uniform scanning	2%	0.3%

Table 2: Breakdown of scan-related commands seen on tracked botnets during the measurement period.

We use two approaches to examine the different growth patterns observed in botnets. First, we plot the cumulative number of unique DNS cache hits for individual botnets over time. Figure 7 shows examples of three predominant growth patterns extracted from our data. As the graphs indicate, footprint growth exhibits different patterns across different botnets. To better understand possible causes behind such patterns, we cross reference each growth pattern with the corresponding inside behavior learned from the IRC tracker. By correlating the two traces, we noticed that botnets with semi-exponential growth patterns (Figure 7.a) exhibit persistent random scanning activity that does not change over time—for example, for one of the botnets the topic of the channel was set to randomly scan port 445 indefinitely, and remained unchanged for over one month. This growth pattern is more closely related to that of worm infections, which is a direct result of their monolithic spreading behavior.

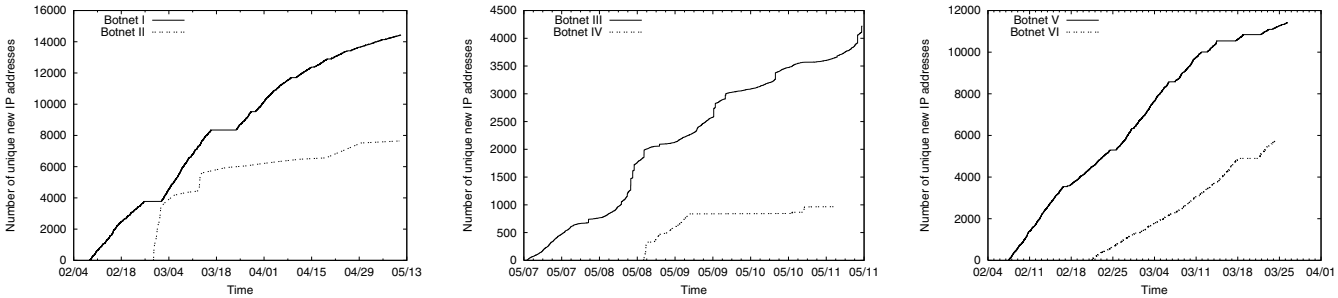
Figure 7.b shows another common pattern representative of botnets with intermittent activity profiles. For example, Botnet III from that figure corresponds to a botnet that infected our honeypots on `3/13/2006`, after which the IRC server went down for the period between `4/12-4/30/2006`. Shortly after the IRC server becomes available, the growth slope drastically increases and our the honeypots were re-infected by the same botnet. Finally, the third growth behavior shown in Figure 7.c was generally observed in botnets using time-scoped scanning commands targeting specific network prefixes as opposed to continuous scanning using localized or random scanning strategies.

To confirm these trends, we also examine botnet growth by leveraging the insider information learned by our tracker. In this case, the results apply to the set of botnets that relay bot message exchanges to the IRC channel. Overall, 52% of the botnets we observed broadcast their messages to the C&C channel. From these



(a) Semi-Exponential (b) Staircase (c) Linear

Figure 7: DNS views showing examples from multiple botnets with the three predominant growth patterns.



(a) Semi-Exponential (b) Staircase (c) Linear

Figure 8: View from IRC tracker showing multiple botnets with the three predominant growth patterns.

messages we estimate the evolution of a botnet by counting the unique sources for messages broadcast to the channel.[7] Figure 8 provides examples of the predominant growth patterns extracted from the tracker logs. For illustrative purposes, we only plot botnets of comparable sizes on a given plot. These trends confirm the heterogeneity observed from the DNS experiments. A closer look at the IRC tracker log shows that, for the most part, these growth patterns are a result of the aforementioned reasons.

4.3 Botnet Structures

Of the 318 total malicious binaries we collected, 60% were IRC bots; only a handful used HTTP for the command and control.[8] Our tracker traces reveal four predominant IRC structures:

- All the bots connect to a single IRC server. This architecture is prevalent among the smaller class of botnets (typically having on the order of a few hundred online users), and it is not uncommon to see such botnets reaching the server's capacity. 70% of the botnets we observed fell into this category.

- By leveraging the server linking capability of the IRC protocol [13], different IRC servers can be connected to form an IRC network supporting large numbers of users. Inferring the exact structure of a bridged botnet, however, can be complicated without explicit information. We infer the presence of bridged structures by examining the server status messages (from the tracker's logs) and seeing whether multiple servers are in use. The discrepancy between local versus total online users can also disclose whether or not bridging is in place. Our analysis shows that 30% of the botnets were bridged on multiple servers, 50% of which were bridged between two servers only. Roughly 25% of the bridged servers were also known public servers.

- Several seemingly unrelated botnets appear, on closer examination, to be very similar when compared in terms of their naming conventions, channel names, and operators' user IDs. In many cases, these botnets seem to belong to the same botmaster(s).

- Lastly, we observe several instances where a selected group of bots were commanded to download an updated binary, which subsequently moved the bots to a different IRC server. We return to this point with an in-depth look at different forms of bot migration in Section 4.7.

4.4 Effective Botnet Sizes

The results of Section 4.2 argue that a botnet's footprint can become fairly large in size (*e.g.*, more than 15,000 bots). That said, the predominant structures we observed were botnets managed by a single or few servers. For that reason, it is doubtful that these structures could support such large numbers of online bots. Therefore, we draw a distinction between a botnet's footprint and the number of bots connected to the IRC channel at a specific time, which we term at the botnet's *effective size*.

Figure 9 plots the number of online bots versus time for several "chatty" servers that broadcast join/leave information for members on the channel. The plots reveal a number of important characteristics, most notably that the maximum size of the online population is (in general) significantly smaller than the botnet's footprint. For

[7]There is of course caveat in using IP address (albeit real or "cloaked" [13]) for this purpose. However, without a better measure of uniqueness at hand, we assume for now that IP addresses are a reasonable measure for uniqueness among members of a single botnet.

[8]We did observe bots that attempted to spread themselves by sending messages to contacts on the compromised machines' IM list, thereby enticing the unsuspecting users to download the malcode.

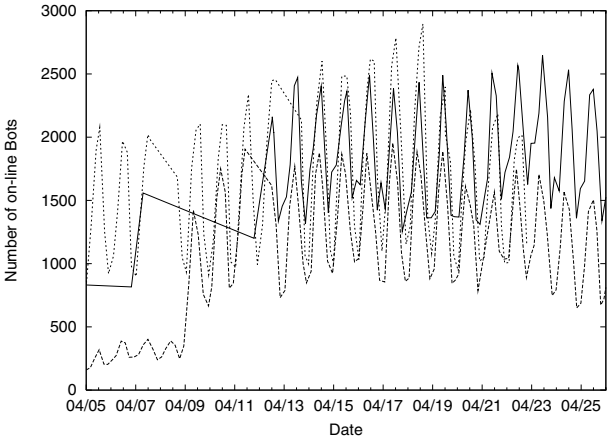


Figure 9: Evolution of effective size for three of the botnets in our study.

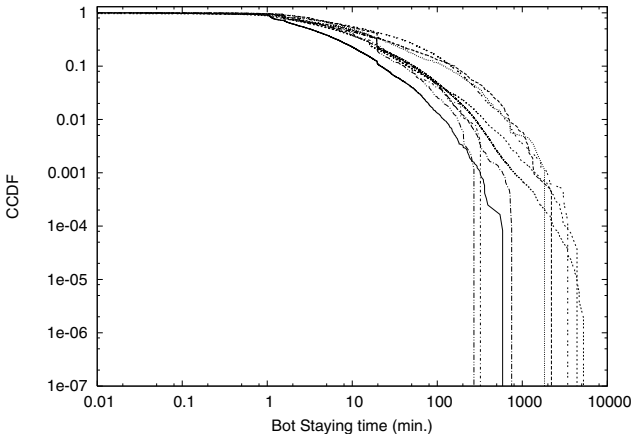


Figure 10: CCDF of bot channel occupancy.

instance, the average footprint for the botnets depicted in Figure 9 was greater than $10,000$, while at most $\approx 3,000$ bots were online at the same time. While the effective size has less impact on long term activities (*i.e.*, executing commands posted as channel topics), it significantly affects the number of minions available to execute timely commands (*e.g.*, DDoS attacks).

Another interesting observation is the strong 24 hour diurnal pattern in the botnet effective size. The notable synchronization in the peaks is the composition of online populations belonging to different time zones. Indeed, this observation confirms the pattern first noted and studied by Dagon *et al.* in their analysis of the connection attempts from Type-I bots to black-holed IRC servers [6].

4.5 Lifetime

The wide discrepancy between the footprint and the effective size is likely due to the relatively long lifetime of a typical botnet. Under these conditions, bot death rates (*e.g.*, as a result of patching) can significantly impact a botnet's effective size. Additionally, as we show shortly, the high churn rate of bots connected to the IRC channel significantly contributes to this disparity.

Figure 10 illustrates the distributions of channel occupancy times for a number of botnets. These results were inferred by tracking unique joins and quits for the IRC servers that broadcast such information. As the graph shows, botnet IRC channels exhibit high churn rates, implying that bots generally do not stay long on the IRC channel. The average staying time for all bots across the botnets we tracked is approximately 25 minutes, with 90% of them staying for less than 50 minutes.

Although we can not explain with certainty what causes this high churn rate, some likely causes include client instability as a result of the infection, machine hibernation, and (as we have frequently seen) botmasters commanding bots to leave the channel. Lastly, inspection of the traces also shows that botmasters have the longest staying times among all users on the channel. Their fascination with keeping a close watch on the activities of the bots under their control, as well as the desire to keep their operator status on the channel, is probably the impetus behind their behavior.

With regards to botnets lifetimes, our data shows that botnets are generally long-lived. Those that were shutdown remained active, on average, for about 47 days before ceasing operation. Among all the botnets we tracked, 84% of the IRC servers were still up at the end of the monitoring period, 55% of which were still actively scanning the Internet. A rather troubling finding is that among the botnets with the longest lifetimes were those that used static IP addresses rather than DNS names for their IRC servers; we tracked three such botnets for nearly three months with footprints exceeding 3,500 bots. Lastly, several botnets were available only intermittently, but as soon as they were re-activated, they quickly regained momentum due to the continual connection attempts of their zombies. These observations raise serious concerns about the effectiveness of current measures in curtailing the botnet problem.

4.6 Botnet Software Taxonomy

Of the 192 confirmed IRC-based bot executables,[9] 138 responded to the probes of the IRC query engine (Phase 2 of the graybox testing process described in Section 3.1). The collected responses were used to assess the threads that the bots start after they are executed. Table 3 tabulates the percentage of binaries that reported threads of each type. For example, almost 50% of the bots run a utility thread that disables anti-virus and firewall processes running on the infected host (the so-called "AV/FW Killer"). Likewise, many bots run an `identd` [12] server which is used to identify the user on the client-end of a TCP connection; this feature is required by some IRC servers and is used to verify that only intended bots join a specific IRC channel. The registry monitor thread checks for modifications to the system's registry and is presumably used to alert the bot of any attempts to disable it.

Utility Software Thread	Frequency (%)
AV/FW Killer	49
Identd Server	43
System Security Monitor	40
Registry Monitor	38

Table 3: The percentage of bots that launched the respective services on the victim machines.

The System Security Monitor is also related to bot self-defense in that it periodically calls the `secure()` function. This function is often called manually by the botmaster to perform rudimentary security "tightening" tasks, such as disabling DCOM services and file sharing. To our surprise, we witnessed newer versions of the secure command that actually patched the LSASS vulnerability.

[9]These binaries were unique based on their MD5 sums.

The number of reported exploit modules bundled within bot binaries varied from 3 to 29, with an average of 15 exploits per binary.[10] The three most popular exploits were `DCOM135`, `LSASS-445`, and `NTPASS`, all of which appear in over 75% of the binaries. We also note that several executables reported identical results, with the exception of their exploit capabilities. This is indicative of the modularity of bot software, in that new exploit modules can be added with relative ease.

OS version	% inf.	Service Pack			
		None	SP1	SP2	SP3+
Win XP	82.6	.47	.52	.01	n/a
Win 2000	16.1	.09	.05	.03	.83
Win Server	1.3	.57	.43	n/a	n/a

Table 4: Distribution of exploited hosts extracted from the IRC tracker logs.

Table 4 indicates that the botnets we tracked target a diverse set of operating systems. It is noteworthy that even the latest version of Windows XP (Service Pack 2) is not immune to attacks. Given the danger that bots pose to end-users, and to the Internet in general, we were interested to see if off-the-shelf anti-virus products offer protection against IRC bots. To answer this question, we subjected all confirmed IRC-bot binaries to virus scans using the Open Source ClamAV tool [4] and Norton's anti-virus, each using their most recent definitions. The results were somewhat reassuring: ClamAV classified 137 of 192 binaries as malicious, and Norton AV detected 179.

Figure 11 depicts the categorization of the 192 binaries based on ClamAV's report[11]. Despite all the binaries having unique MD5 sums, several inspected binaries seem to be logically equivalent. The superfluous differences are mainly due to different configuration parameters (such as contacting a different IRC server or channel, *etc.*) or can be manifestations of polymorphism.

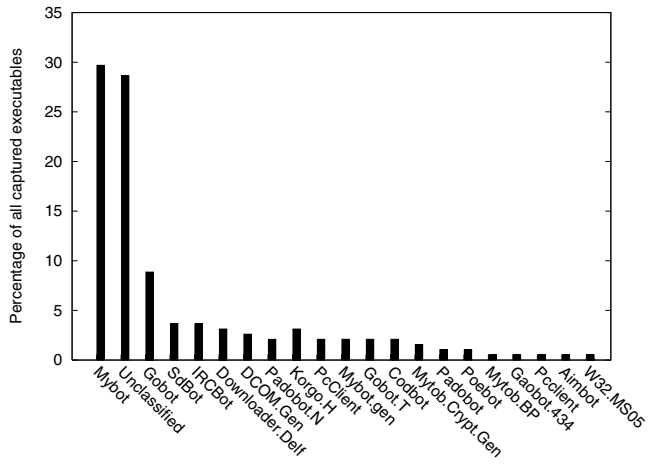


Figure 11: Breakdown of captured IRC-bots based on ClamAV's classification

[10]Many "different" exploits were variants of the same vulnerability, such as the DCOM vulnerability for different ports.

[11]We choose to show results for ClamAV simply because its report was more descriptive compared to Norton AV.

4.7 Insights from the "Insider's view"

The tracker observed botmasters of varying skill levels, ranging from novices frustrated with their lack of success at managing a handful of bots to botmasters performing far more sophisticated behaviors. For example, our traces show that (*i*) botmasters share information regarding what prefixes they should not scan[12], (*ii*) tweak their bots to keep chatter on the C&C channel to a minimum, and (*iii*) actively probe selected bots to detect and isolate "misbehavers" (*i.e.* bots that do not seem to respond to their commands) and similarly, look for "super-bots" with valuable resources (*e.g.*, high bandwidth network links and storage capacities).

Our inside look has also afforded us the opportunity to witness several instances during which bots were commanded to migrate, either by being instructed to move to a new IRC channel/server or to download replacement software that pointed them to a different C&C server. Figure 12 provides a snapshot of one such migration instance captured by the IRC tracker. By simultaneously participating in two separate botnets, the tracker was able to witness the surge in membership in one botnet immediately after a migration command was issued in the other. Additionally, we observed several instances of cloning (*i.e.*, instantiating multiple IRC sessions to a specified server). Many of the cloning events we witnessed can be attributed to attacks intended to overwhelm another IRC server; other events may be evidence of some form of bot "leasing".

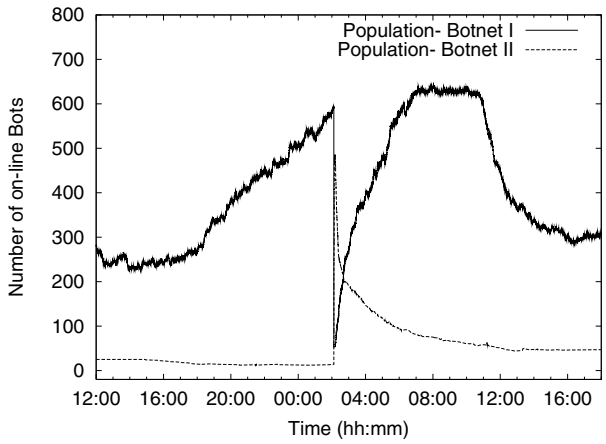


Figure 12: Migration of a botnet as observed by our IRC Tracker.

Command Type	Frequency (%)
Control	33
Scanning	28
Cloning	15
Mining	7
Download	7
Attack	7
Other	3

Table 5: Relative frequency of commands observed across all tracked botnets.

In addition to cloning, the botnets we tracked were used for a variety of purposes (as indicated by the commands issued by their

[12]For example, we observed botmasters alerting each other to avoid scanning certain address prefixes.

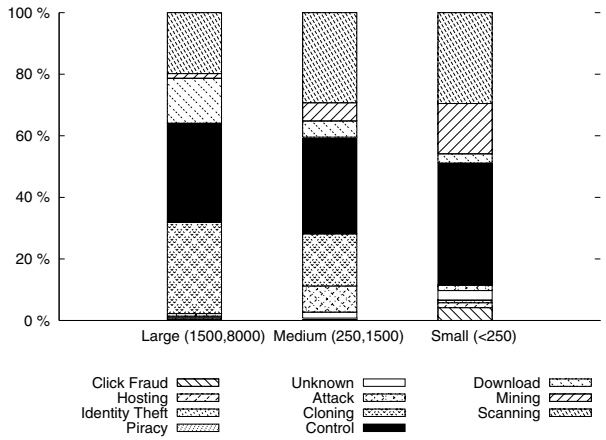


Figure 13: Percentage of command types as a function of observed botnet size.

botmasters). The relative frequency of commands issued across all botnet instances is summarized in Table 5. One can see that in addition to control commands (*e.g.,* channel joins and leaves), scanning commands are relatively popular. The mining category includes commands that collect information about the capabilities of the machines the bots run on (*e.g.,* processor specs, physical memory, etc.), while the attack category includes direct orders by the botmaster for the bots to use their arsenal of flooding-related commands to attack other networked computers.

What is not evident from Table 5 is the diversity in botnet usage. We have observed markedly different patterns of behaviors in the botmasters' personal traits and a difference in the operational utility across different botnets. In the interest of space, we present only how the relative frequency of command types varies across botnets of different sizes. Similar inferences can be made when botnets are grouped according to other criteria.

As Figure 13 illustrates, small botnets (*i.e.*, those with fewer than 250 online nodes) receive a larger portion of control and mining commands. This behavior corresponds to "hands-on" botmasters that devote considerable time and effort to manually control their botnets, something that is possible only for small numbers of machines. On the other hand, medium and large botnets have a larger percentage of cloning and download commands. Cloning can be attributed to the use of a botnet to attack another botnet by overloading its IRC server with a barrage of join requests. This technique is effective only when a considerable number of machines participate in the attack.

5. RELATED WORK

Despite their relatively long presence in the Internet, few formal studies have examined the botnet problem. The Honeynet group [19] was among the first to perform an informal study of the botnet problem. In a related study, Freiling *et al.* [8] presented a proposal for countering certain classes of DDoS attacks originating from botnets through a multi-step process: first bot binaries are collected using honeypots and active responders, then information necessary to join the botnets is extracted by running those binaries on honeypots and allowing them to contact the actual IRC server. Finally, a "silent drone" infiltrates the botnets to collect information that can be useful in dismantling them. Our study is focused on a broader understanding of the botnet phenomenon through a multi-faceted, longitudinal tracking approach that integrates information learned from multiple data collection mechanisms including network traces collected from darknets, DNS cache probes, and "insider" views of botnet activity, made possible by an "active drone" that mimics the behavior of an actual bot learned through graybox testing. This infrastructure is broader both in scale and scope and allows us to draw deeper conclusions about several previously unknown aspects of botnet behavior.

Cooke *et al.* presented an initial look at the prevalence of botnets by measuring the elapsed time before an un-patched system was infected by a botnet [5]. They also highlighted the potential threat from a new generation of botnets that use P2P protocols for their C&C channels. More recently, Barford *et al.* presented an alternative perspective based on an in-depth analysis of bot software source code [3]. Our study complements these efforts by providing multifaceted observations gleaned from real-world botnet behavior and via graybox testing of bot binaries gathered by our distributed collection platform.

Malware collection infrastructures have been the cornerstone of a number of recent initiatives. Indeed, implementing a scalable and stable collection infrastructure is an important research problem in its own right. Vrable *et al.* presented Potemkin, a scalable virtual honeynet system [23]. While Potemkin can be very useful in botnet detection (as it enables malware collection at a large scale), it is inappropriate for long term botnet tracking which requires techniques such as the IRC tracker discussed earlier.

The lightweight responder is a central component in our botnet measurement infrastructure. However, the ability of such responders to faithfully emulate complex, stateful protocols is limited. More recently, Cui *et al.* [24] presented RolePlayer—a protocol independent lightweight responder that tries to overcome some of these limitations by reverting to a real server when the responder fails to produce the proper response. Instead, our infrastructure employs deep interaction honeypots to complement the active responder and capture any sessions it misses. To further reduce the tasks performed by these honeypots, we are currently extending our malware collection system so that the honeypots only handle potential exploit attempts that the lightweight responders can not parse.

Finally, Dagon *et al.* [6] provide an initial analytical model for capturing the spreading behavior of botnets. Their model assumes botnets spreading through uniform scanning. As our results show, this behavior is only exhibited by a narrow class of worm-like botnets, while botnets in general exhibit heterogeneous spreading behavior. Our findings can be used to develop more realistic models that better reflect such behaviors.

6. CONCLUSIONS

Botnets pose one of the most severe threats to the Internet. Despite this fact, our knowledge of botnet behavior is, at best, incomplete. To improve our understanding, we present a composite view that combines measurements from multiple independent sources. Doing so not only produces a richer set of insights, but also allows us to validate the results collected by the different data acquisition methods.

In summary, our results show that botnets are a major contributor to the overall unwanted traffic on the Internet. While botnets' contribution to the aggregate traffic can be mostly attributed to scans used to recruit new victims, botnet scanning behavior is markedly different from that seen by autonomous malware (*e.g.,* worms) because of its manual orchestration. We found that IRC is still the dominant protocol used for C&C communications, and that its use is adapted to satisfy different botmasters' needs. Moreover, the effective sizes of the botnets we studied ranges from a few hundreds

to a few thousands of online bots. On the other hand, botnet footprints are usually much larger than their effective sizes. This discrepancy can be explained by the high churn rate within a botnet; a bot's average channel occupancy is less than half an hour. Finally, our graybox testing technique enabled us to understand the level of sophistication reached by bot software today, which includes self-protection mechanisms and modular packages with multiple attack vectors.

Acknowledgments

This work is supported in part by National Science Foundation grant SCI-0334108. We thank the anonymous reviewers for their insightful comments.

Data Availability

To promote further research and awareness of the botnet problem, the datasets collected from our distributed platform are available to the research community. Additional information on how to get timely access to this data is available at `http://hinrg.cs.jhu.edu/botnets/`.

7. REFERENCES

[1] Paul Baecher, Thorsten Holz, Markus Kötter, and Georg Wicherski. The Malware Collection Tool (mwcollect). Available at `http://www.mwcollect.org/`.

[2] Paul Baecher, Markus Koetter, Thorsten Holz, Maximillian Dornseif, and Felix Freiling. The Nepenthes Platform: An Efficient Approach to Collect Malware. In *Proceedings of the* 9^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sept. 2006.

[3] Paul Barford and Vinod Yagneswaran. An Inside Look at Botnets. To appear in *Series: Advances in Information Security, Springer*, 2006.

[4] Clam AntiVirus. Available at `http://www.clamav.net/`.

[5] Evan Cooke, Farnam Jahanian, and Danny McPherson. The Zombie Roundup: Understanding, Detecting, and Disturbing Botnets. In *Proceedings of the first Workshop on Steps to Reducing Unwanted Traffic on the Internet (STRUTI)* , pages 39–44, July 2005.

[6] David Dagon, Cliff Zou, and Wenke Lee. Modeling Botnet Propagation Using Time Zones. In *Proceedings of the* 13^{th} *Network and Distributed System Security Symposium NDSS*, February 2006.

[7] Edward W. Felten and Michael A. Schneider. Timing attacks on web privacy. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 25–32, New York, NY, USA, 2000. ACM Press.

[8] Felix Freiling, Thorsten Holz, and Georg Wicherski. Botnet Tracking: Exploring a root-cause methodology to prevent denial-of-service attaks. In *Proceedings of* 10^{th} *European Symposium on Research in Computer Security, ESORICS*, pages 319–335, September 2005.

[9] Luis Grangeia. DNS Cache Snooping or Snooping the Cache for Fun and Profit, Available at `http://www.sysvalue.com/papers/DNS-Cache-Snooping/files/DNS_Cache_Snooping_1.1.pdf`, 2004.

[10] Honeyd Virtual Honeypot Framework, `http://www.honeyd.org/`.

[11] IP2LOCATION, Bringing Geography to the Internet. Available at `http://www.ip2location.com/`.

[12] M. St. Johns. RFC 1413: Identification protocol, January 1993.

[13] C. Kalt. Internet Relay Chat: Client Protocol. RFC 2812 (Informational), April 2000.

[14] Dan Kaminsky. Welcome to Planet Sony, `http://www.doxpara.com/`.

[15] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M.Frans Kaashoek. The Click Modular Router. *ACM Transactions on Computer Systems*, 18(3):263–297, 2000.

[16] Willaim Metcalf. Snort In-line. Available at `http://snort-inline.sourceforge.net/`.

[17] Alexandros Ntoulas, Junghoo Cho, and Christopher Olston. What's New on the Web? The Evolution of the Web from a Search Engine Perspective. In *Proceedings of the* 13^{th} *International World Wide Web (WWW) Conference*, pages 1–12, 2004.

[18] Larry Peterson, Tom Anderson, David Culler, and Timothy Roscoe. A Blueprint for Introducing Disruptive Technology into the Internet. *SIGCOMM Computer Communication Reviews*, 33(1):59–64, 2003.

[19] Honeynet Project and Research Alliance. Know your enemy: Tracking Botnets, March 2005. See `http://www.honeynet.org/papers/bots/`.

[20] Niels Provos. A virtual honeypot framework. In *Proceedings of the USENIX Security Symposium*, pages 1–14, August 2004.

[21] Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing I/O Devices on VMware Workstation's Hosted Virtual Machine Monitor. In *USENIX Annual Technical Conference*, 2001. Available at `http://www.vmware.com/`.

[22] The UnrealIRC Team. Unrealircd. See `http://www.unrealircd.com`.

[23] Michael Vrable, Justin Ma, Jay Chen, David Moore, Erik Vandekieft, Alex C. Snoeren, Geoffrey M. Voelker, and Stefan Savage. Scalability, Fidelity and Containment in the Potemkin Virtual Honeyfarm. *Proceedings of ACM SIGOPS Operating System Review*, 39(5):148–162, 2005.

[24] Nick Weaver Weidong Cui, Vern Paxson and Randy H. Katz. Protocol-Independent Adaptive Replay of Application Dialog. In *Proceedings of the* 13^{th} *Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA*, Feb 2006.