Studying Spamming Botnets Using Botlab

John P. John, Alexander Moshchuk, Steven D. Gribble, and Arvind Krishnamurthy
Department of Computer Science & Engineering
University of Washington

Abstract

In this paper we present Botlab, a platform that continually monitors and analyzes the behavior of spam-oriented botnets. Botlab gathers multiple real-time streams of information about botnets taken from distinct perspectives. By combining and analyzing these streams, Botlab can produce accurate, timely, and comprehensive data about spam botnet behavior. Our prototype system integrates information about spam arriving at the University of Washington, outgoing spam generated by captive botnet nodes, and information gleaned from DNS about URLs found within these spam messages.

We describe the design and implementation of Botlab, including the challenges we had to overcome, such as preventing captive nodes from causing harm or thwarting virtual machine detection. Next, we present the results of a detailed measurement study of the behavior of the most active spam botnets. We find that six botnets are responsible for 79% of spam messages arriving at the UW campus. Finally, we present defensive tools that take advantage of the Botlab platform to improve spam filtering and protect users from harmful web sites advertised within botnet-generated spam.

1 Introduction

Spamming botnets are a blight on the Internet. By some estimates, they transmit approximately 85% of the 100+ billion spam messages sent per day [12, 17]. Botnet-generated spam is a nuisance to users, but worse, it can cause significant harm when used to propagate phishing campaigns that steal identities, or to distribute malware to compromise more hosts.

These concerns have prompted academia and industry to analyze spam and spamming botnets. Previous studies have examined spam received by sinkholes and popular web-based mail services to derive spam signatures, determine properties of spam campaigns, and characterize scam hosting infrastructure [1, 31, 32]. This analysis of "incoming" spam feeds provides valuable information on aggregate botnet behavior, but it does not separate activities of individual botnets or provide information on the spammers' latest techniques. Other efforts reverse engineered and infiltrated individual spamming botnets, including Storm [16] and Rustock [3]. However, these techniques are specific to these botnets and their communication methods, and their analysis only considers characteristics of the "outgoing" spam these botnets generate. Honeynets [11, 22, 33] are becoming less applicable to this problem over time, as botnets are increasingly propagating via social engineering and web-based drive-by download attacks that honeynets will not observe. Overall, there is still opportunity to design defensive tools to filter botnet spam, identify and block botnet-hosted malicious sites, and pinpoint which hosts are currently participating in a spamming botnet.

In this paper we turn the tables on spam botnets by using the vast quantities of spam that they generate to monitor and analyze their behavior. To do this, we designed and implemented *Botlab*, a continuously operating botnet monitoring platform that provides real-time information regarding botnet activity. Botlab consumes a feed of all incoming spam arriving at the University of Washington, allowing it to find fresh botnet binaries propagated through spam links. It then executes multiple captive, sandboxed nodes from various botnets, allowing it to observe the precise outgoing spam feeds from these nodes. It scours the spam feeds for URLs, gathers information on scams, and identifies exploit links. Finally, it correlates the incoming and outgoing spam feeds to identify the most active botnets and the set of compromised hosts comprising each botnet.

A key insight behind Botlab is that the combination of *both* incoming and outgoing spam sources is essential for enabling a comprehensive, accurate, and timely analysis of botnet behavior. Incoming spam bootstraps the process of identifying spamming bots, outgoing spam enables us to track the ebbs and flows of botnets' ongoing spam campaigns and establish the ground truth regarding spam templates, and correlation of the two feeds can classify incoming spam according to botnet that is sourcing it, determine the number of hosts active within each botnet, and identify many of these botnet-infected hosts.

1.1 Contributions

Our work offers four novel contributions. First, we tackle many of the challenges involved in building a real-time botnet monitoring platform, including identifying and incorporating new bot variants, and preventing Botlab hosts from being blacklisted by botnet operators.

Second, we have designed network sandboxing mechanisms that prevent captive bot nodes from causing harm, while still enabling our research to be effective. As well, we discuss the long-term tension between effectiveness and safety in botnet research given botnets' trends, and we present thought experiments that suggest that a determined adversary could make it extremely difficult to conduct future botnet research in a safe manner.

Third, we present interesting behavioral characteristics of spamming botnets derived from our multi-perspective analysis. For example, we show that just a handful of botnets are responsible for most spam received by UW, and attribute incoming spam to specific botnets. As well, we show that the bots we analyze use simple methods for locating their C&C servers; if these servers were efficiently located and shut down, much of today's spam flow would be disrupted. As another example, in contrast to earlier findings [32], we observe that some spam campaigns utilize multiple botnets.

Fourth, we have implemented several prototype defensive tools that take advantage of the real-time information provided by the Botlab platform. We have constructed a Firefox plugin that protects users from scam and phishing web sites propagated by spam botnets. The plug-in blocked 40,270 malicious links emanating from one botnet monitored by Botlab; in contrast, two blacklist-based defenses failed to detect any of these links. As well, we have designed and implemented a Thunderbird plugin that filters botnet-generated spam. For one user, the plugin reduced the amount of spam that bypassed his SpamAssassin filters by 76%.

The rest of this paper is organized as follows. Section 2 provides background material on the botnet threat. Section 3 discusses the design and implementation of Botlab. We evaluate Botlab in Section 4 and describe applications we have built using it in Section 5. We discuss our thoughts on the long-term viability of safe botnet research in Section 6. We present related work in Section 7 and conclude in Section 8.

2 Background on the Botnet Threat

A botnet is a large-scale, coordinated network of computers, each of which executes specific bot software. Botnet operators recruit new nodes by commandeering victim hosts and surreptitiously installing bot code onto them; the resulting army of "zombie" computers is typically controlled by one or more command-and-control (C&C) servers. Botnet operators employ their botnets to send spam, scan for new victims, steal confidential information from users, perform DDoS attacks, host web servers and phishing content, and propagate updates to the botnet software itself.

Botnets originated as simple extensions to existing Internet Relay Chat (IRC) softbots. Efforts to combat botnets have grown, but so has the demand for their services. In response, botnets have become more sophisticated and complex in how they recruit new victims and mask their presence from detection systems:

Propagation: Malware authors are increasingly relying on social engineering to find and compromise victims, such as by spamming users with personal greeting card ads or false upgrade notices that entice them to install malware. As propagation techniques move up the protocol stacks, the weakest link in the botnet defence chain becomes the human user. As well, systems such as passive honeynets become ineffective at detecting new botnet software, instead requiring active steps to gather and classify potential malware.

Customized C&C protocols: While many of the older botnet designs used IRC to communicate with C&C servers, newer botnets use encrypted and customized protocols for disseminating commands and directing bots [5, 7, 26, 29]. For example, some botnets communicate via HTTP requests and responses carrying encrypted C&C data. Manual reverse-engineering of bot behavior has thus become time-consuming if not impossible.

Rapid evolution: To evade detection from trackers and anti-malware software, some newer botnets morph rapidly. For instance, most malware binaries are often packed using polymorphic packers that generate different looking binaries even though the underlying code base has not changed [23]. Also, botnet operators are also moving away from relying on a single web server to host their scams, and instead are using *fast flux DNS* [10]. In this scheme, attackers rapidly rebind the server DNS name to different botnet IP addresses, in order to defend against IP blacklisting or manual server take-down. Finally, botnets also make updates to their C&C protocols, by incorporating new forms of encryption and command distribution.

Moving forward, analysis and defense systems must contend with the increasing sophistication of botnets. Monitoring systems must be pro-active in collecting and executing botnet samples, as botnets and their behavior change rapidly. As well, botnet analysis systems will increasingly have to rely on external observations of botnet behavior, rather than necessarily being able to crack and reverse engineer botnet control traffic.

3 The Botlab Monitoring Platform

The Botlab platform produces fresh information about spam-oriented botnets, including their current campaigns, constituent bots, and C&C servers. Botlab partially automates many aspects of botnet monitoring, reducing but not eliminating the manual effort required of a human operator to analyze new bot binaries and incorporate them into Botlab platform.

Botlab's design was motivated by four requirements:

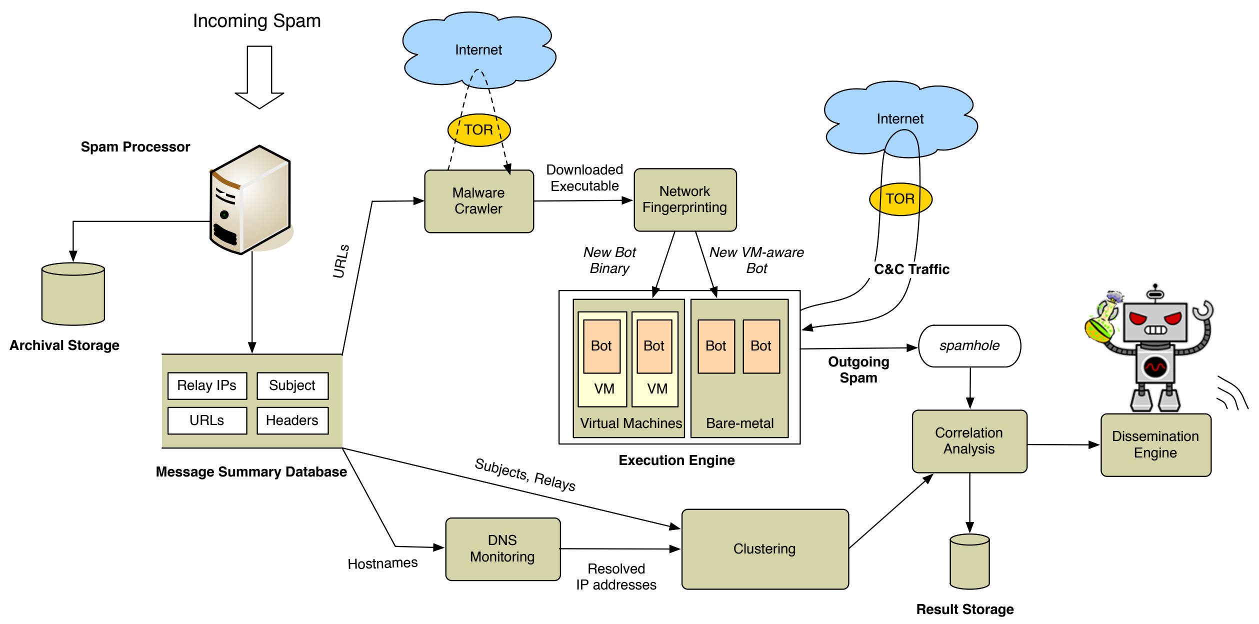

Figure 1: **Botlab Architecture.** Botlab coordinates and monitors multiple source of data about spam botnets, including incoming spam from the University of Washington, and outgoing spam generated by captive bot nodes.

1. *Attribution:* Botlab must identify the spam botnets that are responsible for campaigns and the hosts that belong to those botnets.
2. *Adaptation:* Botlab must track changes in the botnets' behavior over time.
3. *Immediacy:* Because the value of information about botnet behavior degrades quickly, Botlab must produce information on-the-fly.
4. *Safety:* Botlab must not cause harm.

There is a key tension in our work between safety and effectiveness, similar to tradeoff between safety and fidelity identified in the Potemkin honeyfarm [27]. We considered and rejected several mechanisms that would improve the quality of information we gather, but that could potentially let a captive botnet node cause harm to external hosts or users. In Section 6, we discuss this tension in more detail and comment on the long-term viability of safe botnet research.

Figure 1 shows the Botlab architecture. We now describe Botlab's main components and techniques.

3.1 Incoming Spam

Botlab monitors a live feed of spam received by approximately 200,000 University of Washington e-mail addresses. On average, UW receives 2.5 million e-mail messages each day, over 90% of which is classified as spam. We use this spam feed to collect new malware binaries, described next, and within Botlab's correlation engine, described in Section 3.5.

3.2 Malware Collection

Running captive bot nodes requires up-to-date bot binaries. Botlab obtains these in two ways. First, many botnets spread by emailing malicious links to victims; accordingly, Botlab crawls URLs found in its incoming spam feed. We typically find approximately 100,000 unique URLs per day in our spam feed, 1% of which point to malicious executables or drive-by downloads. Second, Botlab periodically crawls binaries or URLs contained in public malware repositories [2, 21] or collected by the MWCollect Alliance honeypots [18].

Given these binaries, a human operator then uses Botlab's automated tools for malware analysis and fingerprinting to find bot binaries that actively send spam, as discussed next in section 3.3. Our experience to date has yielded two interesting observations. First, though the honeypots produced about 2,000 unique binaries over a two month period, none of these binaries seem to be spamming bots. A significant fraction of the honeypot binaries were traditional IRC-based bots, whereas the spamming binaries we identified from other sources all used non-IRC protocols. This suggests that spamming bots propagate through social engineering techniques, rather than the automated compromise of remote hosts.

Second, many of the malicious URLs seen in spam point to legitimate web servers that have been hacked to provide malware hosting. Since malicious pages are typically not linked from the legitimate pages on these web servers, an ordinary web crawl will not find them.

3.3 Identifying Spamming Bots

Botlab executes spamming bots within sandboxes to monitor botnet behavior. However, we must first prune the binaries obtained by Botlab to identify those that correspond to spamming bots and to discard any duplicate binaries.

Simple hashing is insufficient to find *all* duplicates, as malware authors frequently repack binaries or release slightly modified versions to circumvent signature-based security tools. Relying on anti-virus software is also impractical, as these tools do not detect many new malware variants.

To obtain a more reliable behavioral signature, Botlab produces a *network fingerprint* for each binary it considers. A network fingerprint captures information about the network connections initiated by a binary. To obtain it, we execute each binary in a safe sandbox and log all outbound network connection attempts. A network fingerprint will then consist of a set of flow records of the form `<protocol, IP address, DNS address, port>`. Note that the DNS address field might be blank if a bot communicates with an IP directly, instead of doing a DNS lookup.

Once network activity is logged, we extract the flow records. We execute each binary two times and take the network fingerprint to be the set of flow records which are common across both executions. This eliminates any random connections which do not constitute stable behavioral attributes. For example, some binaries harvest e-mail addresses by searching `google.com` for random search terms, and following links to the highest-ranked search results; repeated execution identifies and discards these essentially random connection attempts.

Given the network fingerprints N_1 and N_2, of two binaries B_1 and B_2 respectively, we define the similarity coefficient of the binaries, $S(B_1, B_2)$, to be:

$$S(B_1, B_2) = \frac{|N_1 \cap N_2|}{|N_1 \cup N_2|}$$

If the similarity coefficient of two binaries is sufficiently high (we use 0.2 as the threshold), we consider the binaries to be behavioral duplicates. As well, binaries which attempt to send e-mail are classified as spamming bots.

We took a step to validate our duplicate elimination procedure. Unfortunately, given a pair of arbitrary binaries, determining that they are behavioral duplicates is undecidable, so we must rely on an approximation. For this, we used five commercial anti-virus tools and a set of 192 malicious binaries for which these tools had signatures. We considered a pair of binaries to be duplicates if their anti-virus tags matched in the majority of five tools. Network fingerprinting matched this tag-based classification 97% of the time, giving us reasonable confidence in its ability to detect duplicates. Note again that anti-virus tools lack signatures for many new binaries our crawler analyzes, making them unfit to use as our main duplicate suppression method.

3.3.1 Safely generating fingerprints

The tension between safety and effectiveness is particularly evident when constructing signatures of newly gathered binaries. A safe approach would log emitted network packets, but drop them instead of transmitting them externally; unfortunately, this approach is ineffective, since many binaries must first communicate with a C&C server or successfully transmit probe email messages before fully activating. An effective approach would allow a binary unfettered access to the Internet; unfortunately, this would be unsafe, as malicious binaries may perform DoS attacks, probe or exploit remote vulnerabilities, transmit spam, or relay botnet control traffic.

Botlab attempts to walk the tightrope between safety and effectiveness. We provide a human operator with tools that act as a safety net: traffic destined to privileged ports, or ports associated with known vulnerabilities, is automatically dropped, and limits are enforced on connections rates, data transmission, and the total window of time in which we allow a binary to execute. As well, Botlab provides operators with the ability to redirect outgoing SMTP traffic to *spamhole*, an emulated SMTP server that traps messages while fooling the sender into believing the message was sent successfully.

Our safety precautions are strict, and we are very confident that our research to date has been safe. However, the transmission of any network traffic poses some degree of risk of causing harm to the receiver, particularly when the traffic originates from an untrusted binary downloaded from the Internet. In section 6, we present our thoughts on the long-term viability of safely conducting this research.

3.3.2 Experience classifying bots

We have found that certain bots detect when they are being run in a virtual machine and disable themselves. To identify VMM detection, Botlab generates two network fingerprints for each binary: we execute the binary in a VMware virtual machine and also on a bare-metal machine containing a fresh Windows installation. By comparing the resulting two network fingerprints, we can infer whether the binary is performing any VM detection.

Some of the spamming binaries we analyzed made initial SMTP connections, but subsequently refused to send spam. For example, one spam bot connected to spamhole, but never sent any spam messages after receiving the initial greeting string from the SMTP server. We deduced that this bot was checking that the greeting string included the domain name to which the bot was connecting, and we modified spamhole to return appropriate domain names in the string.

We also observed that some spam bots perform more sophisticated SMTP verification before they send spam. For example, when the MegaD bot begins executing, it transmits a test e-mail to a special MegaD mail server, verifying each header it receives during the SMTP handshake. MegaD's mail server returns a message ID string after sending the message, which the bot then sends to its C&C server. The C&C server verifies that the message with this ID was actually delivered to the MegaD mail server before giving any further instructions to the bot. Accordingly, to generate a signature for MegaD, and later, to continuously execute a captured MegaD node, the human operator had to indicate to Botlab to deflect SMTP messages destined for MegaD's mail server from the spamhole to the live Internet.

Some bots do not send spam through SMTP, but instead use HTTP-based web services. For example, a Rustock variant rotates through valid `hotmail.com` accounts to transmit spam. To safely intercept this spam, we had to construct infrastructure that spoofs Hotmail's login and mail transmission process, including using fake SSL certificates during login. Fortunately, this variant does not check the SSL certificates for validity. However, if the variant evolves and validates the certificate, we would not be able to safely analyze it.

3.4 Execution Engine

Botlab executes spamming bot binaries within its execution engine. The engine runs each bot within a VM or on a dedicated bare-metal box, depending on whether the bot binary performs VMM detection. In either case, Botlab sandboxes network traffic to prevent harm to external hosts. We re-use the network safeguards described in the previous section in the execution engine sandbox: our sandbox redirects outgoing e-mail to spamhole, permits only traffic patterns previously identified as safe by a human operator to be transmitted to the Internet, and drops all other packets. Traffic permitted on the Internet is also subject to the same rate limiting policies we previously described.

Though we have analyzed thousands of malware binaries to date, only a surprisingly small fraction correspond to unique spamming botnets. In fact, we have so far found just seven spamming bots: Grum, Kraken, MegaD, Pushdo, Rustock, Srizbi, and Storm [1]. We believe these are the most prominent spam botnets existing today, and our results suggest that they are responsible for sending most of the world's spam. Thus, it appears the spam botnet landscape consists of just a handful of key players.

[1]We derived botnet names according to tags with which anti-virus tools classify the corresponding binaries.

3.4.1 Avoiding blacklisting

If the botnet owners learn about Botlab's existence, they might attempt to blacklist IP addresses belonging to the University of Washington. The C&C servers would then refuse connections to Botlab's captive bots, rendering Botlab ineffective. To prevent this, Botlab routes any bot traffic permitted onto the Internet, including C&C traffic, through the anonymizing Tor network [4]. Our malware crawler is also routed through Tor.

Some bots track and report the percentage of e-mail messages successfully sent and e-mail addresses for which sending failed. These lists can be used by botnet owners to filter out invalid or outdated addresses. To avoid detection, we had to ensure that our bots did not report 100% delivery rates, as these are unlikely to happen in the real world. Doing so was easy; our bots experience many send errors because of failed DNS lookups for mail servers. Thus, we simply rely on DNS to provide us with a source of randomness in bot-reported statistics. Should bot masters begin to perform a more complicated statistics analysis, more controlled techniques for introducing random failures in spamhole might become necessary.

3.4.2 Multiple C&C servers

Some botnets partition their bots across several C&C servers. For example, in Srizbi, different C&C servers are responsible for sending different classes of spam. These spam classes differ in subject line, content, embedded URLs, and even languages. If we were to run only a single Srizbi bot binary, it would connect to one C&C server, and therefore we would only have a partial view of the overall botnet activity.

To rectify this, we take advantage of a C&C redundancy mechanism built into many bots, including Srizbi: if the primary C&C server goes down, an alternate C&C server is selected either via hardcoded IP addresses or programmatic DNS lookups. Botlab can thus block the primary C&C server(s) and learn additional C&C addresses. Botlab can then run multiple instances of the same bot, each routed to a different C&C server.

3.5 Correlating incoming and outgoing spam

Botlab's correlation analyzer combines our different sources of botnet information to provide a more complete view into overall botnet activity. For example, armed with a real-time *outgoing* spam feed, we can classify spam received by our *incoming* spam feed according to the botnet that is responsible for sending it. We will describe how we derived our classification algorithm and evaluate its accuracy in Section 4.3.1.

For spam that cannot be attributed to a particular botnet using our correlation analysis, we use *clustering* analysis to identify sets of relays used in the same spam campaign. In Section 4.2, we evaluate various ways in which this clustering can be performed. If there is a significant overlap between a campaign's relay cluster and known members of a particular botnet (where botnet membership information is derived from the earlier correlation analysis), then we can merge the two sets of relays to derive a more complete view of botnet membership.

3.6 Summary

We have outlined an architecture for Botlab, a real-time spam botnet monitoring system. Some elements of Botlab have been proposed elsewhere; our principal contribution is to assemble these ideas into an end-to-end system that can safely identify malicious binaries, remove duplicates, and execute them without being blacklisted. By correlating the activity of captured bots with the aggregate incoming spam feed, the system has the potential to provide more comprehensive information on spamming botnets and also enable effective defenses against them. We discuss these issues in the remainder of the paper.

4 Analysis

We now present an analysis of botnets that is enabled by our monitoring infrastructure. First, we examine the actions of the bots being run in Botlab, characterize their behavior, and analyze the properties of the outgoing spam feed they produce. Second, we analyze our incoming spam feed to extract coarse-grained, aggregate information regarding the perpetrators of malicious activity. Finally, we present analysis that is made possible by studying both the outgoing and incoming spam feeds. Our study reveals several interesting aspects of spamming botnets.

4.1 The Spam Botnets

In our analysis, we focus on seven spam botnets: Grum, Kraken, MegaD, Pushdo, Rustock, Srizbi, and Storm. Although our malware crawler analyzed thousands of potential executables, after network fingerprinting and pruning described earlier, we found that only variants of these seven bots actively send spam. Next, we summarize various characteristics of these botnets and our experience running them.

4.1.1 Behavioral Characteristics

Table 1 summarizes various characteristics of our botnets, which we have monitored during the past six months. The second column depicts the number of days on which we have observed a botnet actively sending spam. We found that keeping all botnets active simultaneously is difficult. First, locating a working binary for each botnet required vastly different amounts of time, depending on the timings of botnet propagation campaigns. For example, we have only recently discovered Grum, a new spamming botnet which has only been active for 8 days, whereas Rustock has been running for more than 5 months. Second, many bots frequently go offline for several days, as C&C servers are taken down by law enforcement, forcing the bot herders to re-establish new C&C servers. Sometimes this breaks the bot binary, causing a period of inactivity until a newer, working version is found.

The amount of outgoing spam an individual bot can generate is vastly different across botnets. MegaD and Srizbi bots are the most egregious: they can send out more than 1,500 messages per minute, using as many as 80 parallel connections at a time, and appear to be limited only by the client's bandwidth. On the other hand, Rustock and Storm are "polite" to the victim – they send messages at a slow and constant rate and are unlikely to saturate the victim's network connection. Big variability in send rates suggests these rates might be useful in fingerprinting and distinguishing various botnets.

Bots use various methods to locate and communicate with their C&C servers. We found that many botnets use very simple schemes. Rustock, Srizbi, and Pushdo simply hardcode the C&C's IP address in the bot binary, and MegaD hardcodes a DNS name. Kraken uses a proprietary algorithm to generate a sequence of dynamic DNS names, which it then attempts to resolve until it finds a working name. An attacker registers the C&C server at one of these names and can freely move the C&C to another name in the event of a compromise. In all of these cases, Botlab can efficiently pinpoint the IP addresses of the active botnet C&C servers; if these servers could be efficiently located and shut down, the amount of worldwide spam generated would be substantially reduced.

Botnet	# days active in trace	total spam messages	spam send rate (messages/min)	C&C protocol	C&C servers contacted over lifetime	C&C discovery
Grum	8 days	864,316	344	encrypted HTTP, port 80	1	static IP (206.51.231.192)
Kraken	25 days	5,046,803	331	encrypted HTTP, port 80	41	algorithmic DNS lookups
Pushdo	59 days	4,932,340	289	encrypted HTTP, port 80	96	set of static IPs
Rustock	164 days	7,174,084	33	encrypted HTTP, port 80	1	static IP (208.72.169.54)
MegaD	113 days	198,799,848	1638	encrypted custom protocol, ports 80 and 443	21	static DNS name (majzufaiuq.info)
Srizbi	51 days	86,003,889	1848	unencrypted HTTP, port 4099	20	set of static IPs
Storm	50 days	961,086	20	p2p (Overnet)	N/A	p2p

Table 1: **The botnets monitored in Botlab.** Table gives characteristics of representative bots participating in the seven botnets. Some bots use all available bandwidth to send more than a thousand messages per minute, while others are rate-limited. Most botnets use HTTP for C&C communication. Some do not ever change the C&C server address yet stay functional for a long time.

Although recent analysis suggests that botnet control is shifting to complicated decentralized protocols as exemplified by Storm [16, 26], we found the majority of our other spam bots use HTTP to communicate with their C&C server. Using HTTP is simple but effective, since bot traffic is difficult to filter from legitimate web traffic. HTTP also yields a simple pull-based model for botnet operators: a new bot makes an HTTP request for work and receives an HTTP response that defines the next task. Upon completing the task, the bot makes another request to relay statistics, such as valid and invalid destination addresses, to the bot master. All of our HTTP bots follow this pattern, which is easier to use and appears just as sustainable as a decentralized C&C protocol such as Storm's protocol.

We checked whether botnets frequently change their C&C server to evade detection or reestablish a compromised server. The column "C&C servers contacted" of Table 1 shows how many times a C&C server used by a bot was changed. Surprisingly, many bots change C&C servers very infrequently; for example, the various copies of Rustock and Srizbi bots have used the same C&C IP address for 164 and 51 days, respectively, and experienced no downtime during these periods. Some bots are distributed as a set of binaries, each with different hardcoded C&C information. For example, we found 20 variants of Srizbi, each using one hardcoded C&C IP address. The C&C changes are often confined to a particular subnet; the 10 most active /16 subnets contributed 103 (57%) of all C&C botnet servers we've seen. As well, although none of the botnets shared a C&C server, we found multiple overlaps in the corresponding subnets; one subnet (208.72.*.*) provided C&C servers for Srizbi, Rustock, Pushdo, and MegaD, suggesting infrastructural ties across different botnets.

4.1.2 Outgoing Spam Feeds

The spam generated by our botnets is a rich source of information regarding their malicious activities. The content of the spam emails can be used to identify the scams perpetrated by the botnets (as discussed in Section 4.3) and help develop application-level defenses for end-hosts (see Section 5). In this section, we analyze the characteristics of the spam mailing lists, discuss the reach of various botnets, and examine whether spam subjects could be used as fingerprints for the botnets.

Size of mailing lists: We first use the outgoing spam feeds to estimate the size of the botnets' recipient lists. We assume the following model of botnet behavior:

- A bot periodically obtains a new chunk of recipients from the master and sends spam to this recipient list. Let c be the chunk size.
- On each such request, the chunk of recipients is selected uniformly at random from the spam list.
- The chunk of recipients received by a bot is much smaller than the spam list size N.

Assuming these are true, the probability of a particular email address from the spamlist appearing in k chunks of recipients obtained by a bot is $1 - (1 - c/N)^k$. As the second term decays with k, the spam feed will expose the entire recipient list in an asymptotic manner, and eventually most newly-picked addresses will be duplicates of previous picks. Further, if we recorded the first m recipient addresses from a spam trace, the expected number of repetitions of these addresses within the next k chunks is $m[1 - (1 - c/N)^k]$.

We have observed that MegaD, Rustock, and Kraken follow this model. We fit the rates at which they see duplicates in their recipient lists into the model above to obtain their approximate spam list sizes. We estimate

MegaD's spam list size to be *850 million addresses*, Rustock's to be *550 million*, and Kraken's *350 million*. Srizbi and Pushdo partition their spam lists in a way that prevents the above analysis, and we have not yet collected enough data for Grum and Storm to reliably estimate their spam list sizes.

	MegaD	Kraken	Rustock
Kraken	28%	N/A	7%
MegaD	N/A	8%	9%
Pushdo	0%	0%	0%
Rustock	15%	6%	N/A
Srizbi	21%	10%	8%
Storm	24%	11%	7%

Table 2: **Overlap between recipient spam lists.** The table shows the fraction of each botnet's recipient list that is shared with MegaD, Kraken, and Rustock's recipient lists. For example, Kraken shares 28% of its recipient list with MegaD.

Overlap in mailing lists: We next examine whether botnets systematically share parts of their spam lists. This analysis is similar to that of spamlist size. For instance, if m_A recipients have been observed in the trace of botnet A, and if f_{AB} is the fraction of this recipient list that is shared with botnet B, then the expected number of occurrences of the m_A addresses in a random trace of B's spam feed is $\textit{shared}_{AB} = f_{AB} \cdot m_A[1-(1-c_B/N_B)^k]$.

We can solve this equation for f_{AB} as long as we know the chunk size c_B, the observed address overlap $\textit{shared}_{AB}$, and the spam-list size N_B. We have found c_B and $\textit{shared}_{AB}$ for all botnets, and plugged in the N_B values for MegaD, Rustock, and Kraken, which we found above. We show the recipient list overlaps for these three botnets in Table 2.

All botnets except Pushdo share a significant number of their addresses with MegaD. For example, Kraken shares more than a fourth of its mailing list with MegaD. The fractions shared with Kraken and Rustock are smaller and stay within 11%. Most of this sharing is likely due to a small set of easily harvested e-mail addresses, such as addresses posted on web sites in clear-text, that have been collected by more than one botnet. However, it is also possible that botnets such as Kraken, Storm, and Srizbi somehow obtained a sizable portion of MegaD's spamlist database in order to seed or expand their own lists. Pushdo does not share any addresses with any of the bots, suggesting it is using a proprietary method of obtaining its recipient list. These results suggest that spammers can benefit from using multiple botnets to get wider reach, a behavior that we in fact observe and discuss in Section 4.3.

Spam subjects: Botnets carefully design and hand-tune custom spam subjects to defeat spam filters and attract attention. We have found that between any two spam botnets, there is *no* overlap in subjects sent within a given day, and an average overlap of 0.3% during the length of our study. This suggests that subjects are useful for classifying spam messages as being sent by a particular botnet. To apply subject-based classification, we remove any overlapping subjects, leaving, on average, 489 subjects per botnet on a given day. As well, a small number of subjects include usernames or random message IDs. We remove these elements and replace them with regexps using an algorithm similar to AutoRE [31]. We will evaluate and validate this classification scheme using our incoming spam in Section 4.3.1.

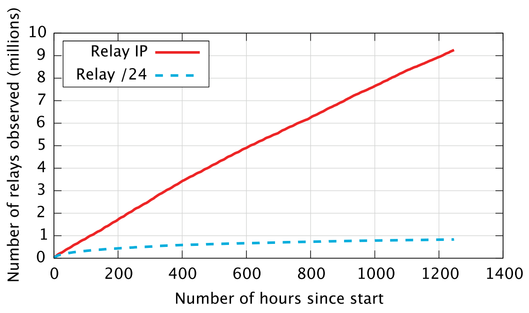

Figure 2: **Number of distinct relay IPs and the /24s containing them.**

4.2 Analysis of Incoming Spam

We analyze 46 million spam messages obtained from a 50-day trace of spam from University of Washington and use it to characterize the hosts sending the spam, the scam campaigns propagated using spam, and the web hosting infrastructure for the scams. To do this, each spam message is analyzed to extract the set of relays through which the purported sender forwarded the email, the subject, the recipient address, other SMTP headers present in the email, and the various URLs embedded inside the spam body.

We found that on average, 89.2% of the incoming mail at UW is classified as spam by UW's filtering systems. Around 0.5% of spam contain viruses as attachments. Around 95% of the spam messages contain HTTP links, and 1% contain links to executables.

4.2.1 Spam sources

Figure 2 plots the total number of distinct last-hop relays seen in spam messages over time. We consider only the IP of the last relay used before a message reaches UW's mail servers, as senders can spoof other relays. The number of distinct relay IPs increases steadily over time and reaches 9.5 million after 7 weeks worth of spam messages. Two factors could be responsible for keeping this

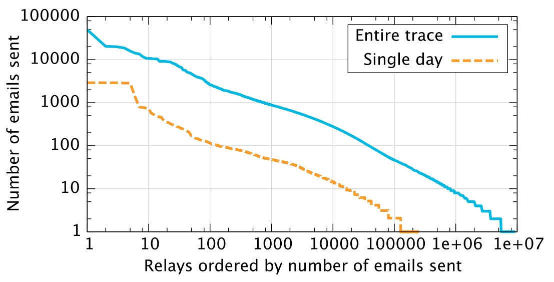

Figure 3: **Number of messages sourced by distinct relay IPs, over a single day and the entire trace.**

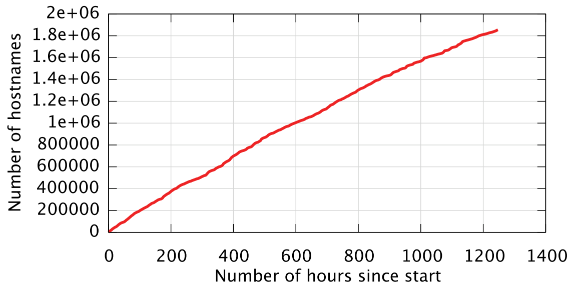

Figure 4: **Number of distinct hostnames in URLs conveyed by spam.** Spammers constantly register new DNS names.

growth linear. One is a constant balance between the influx of newly-infected bots and the disappearance of disinfected hosts. Another is the use of dynamic IP (DHCP) leases for end hosts, which causes the same physical machine to manifest itself under different IPs. To place a lower bound on the number of distinct spam *hosts* given the DHCP effect, Figure 2 also shows the number of distinct /24's corresponding to spam relays, assuming that the IPs assigned by DHCP to a particular host stay in the /24 range. The constantly changing list of IPs relaying spam does indicate that simple IP-based blacklists will not be very effective at identifying spam.

Figure 3 graphs the number of messages each distinct relay has sent during our trace. We also show the number of messages sent by each relay on a particular day, where DHCP effects are less likely to be manifested. On any given day, only a few tens of relays send more than 1,000 spam messages, with the bulk of the spam conveyed by the long tail. In fact, the relays that sent over 100 messages account for only 10% of the spam, and the median number of spam messages per relay is 6. One could classify the heavy hitters as either well-known open mail relays or heavily provisioned spam pumps operated by miscreants. We conjecture that most of the long tail corresponds to compromised machines running various kinds of bots.

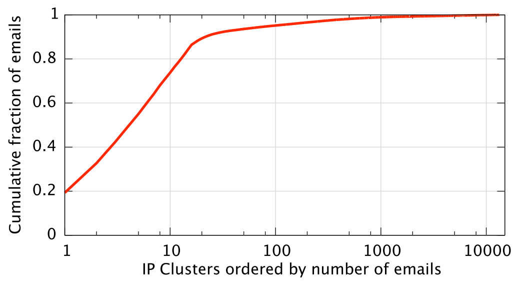

Figure 5: **Clustering spam messages by the IP of URLs contained within them.** Links in 80% of spam point to only 15 distinct IP clusters.

4.2.2 Spam campaigns and Web hosting

We next examine whether we can identify and characterize individual spam campaigns based on our incoming spam. Ideally, we would cluster messages based on similar content; however, this is difficult as spammers use sophisticated content obfuscation to evade spam detection. Fortunately, more than 95% of spam in our feed contains links. We thus cluster spam based on the following attributes: 1) the domain names appearing in the URLs found in spam, 2) the content of Web pages linked to by the URLs, and 3) the resolved IP addresses of the machines hosting this content. We find that the second attribute is the most useful for characterizing campaigns.

Clustering with URL domain names revealed that for any particular day, 10% of the observed domain names account for 90% of the spam. By plotting the number of distinct domain names observed in our spam feed over time (shown in Figure 4), we found that the number of distinct hostnames is large and increases steadily, as spammers typically use newly-registered domains. (In fact, on average, domain names appearing in our spam are only two weeks old based on `whois` data.) Consequently, domain-based clustering is too fine-grained to reveal the true extent of botnet infections.

Our content clustering is performed by fetching the Web page content of all links seen in our incoming spam. We found that nearly 80% of spam pointed to just 11 distinct Web pages, and the content of these pages did not change during our study. We conclude that while spammers try to obfuscate the content of messages they send out, the Web pages being advertised are static. Although this clustering can identify distinct campaigns, it cannot accurately attribute them to specific botnets. We revisit this clustering method in Section 4.3.2, where we add information about our botnets' outgoing spam.

For IP-based clustering, we analyzed spam messages collected during the last week of our trace. We extracted hostnames from all spam URLs and performed DNS lookups on them. We then collected sets of resolved IPs from each lookup, merging any sets sharing

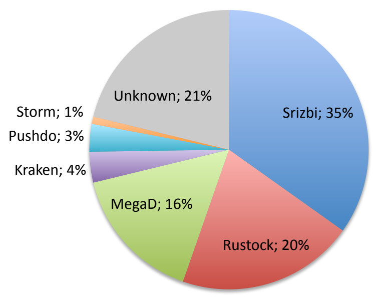

Figure 6: **Average contributions of each botnet to incoming spam received at University of Washington.** 79% of spam comes from six spam botnets, and 35% comes from just one botnet, Srizbi.

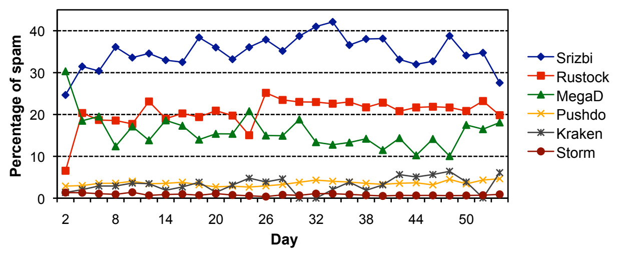

Figure 7: **Breakdown of spam e-mails by botnet over time.** Most botnets contribute approximately the same fraction of spam to our feed over our study period, with Srizbi, Rustock, and MegaD being the top contributors. Kraken shows gaps in activity on days 28-32 and 52. Day 1 corresponds to March 13, 2008.

a common IP. Finally, we grouped spam messages based on these IP clusters; Figure 5 shows the result. We found that 80% of the spam corresponds to the top 15 IP clusters (containing a total of 57 IPs). In some cases, the same Web server varied content based on the domain name that was used to access it. For example, a single server in Korea hosted 20 different portals, with demultiplexing performed using the domain name. We conjecture that such Web hosting services are simultaneously supporting a number of different spam campaigns. As a consequence, web-server-based clustering is too coarse-grained to disambiguate individual botnets.

4.3 Correlation Analyses

We now bring together two of our data sources, our outgoing and incoming spam feeds, and perform various kinds of correlation analyses [2], including: 1) classifying spam according to which botnet sourced it, 2) identifying spam campaigns and analyzing botnet partitioning, 3) classifying and analyzing spam used for recruiting new victims, and 4) estimating botnet sizes and producing botnet membership lists.

4.3.1 Spam classification

To classify each spam message received by University of Washington as coming from a particular botnet, we use subject-based signatures we derived in Section 4.1.2. Each signature is *dynamic* — it changes whenever botnets change their outgoing spam. We have applied these signatures to a 50-day trace of incoming spam messages received at University of Washington in March and April 2008. Figure 6 shows how much each botnet contributed to UW spam on average, and Figure 7 shows how the breakdown behaves over time. We find that on average, our six botnets were responsible for 79% of UW's incoming spam. This is a key observation: it appears that for spam botnets, only a handful of major botnets produce most of today's spam. In fact, 35% of all spam is produced by just *one* botnet, Srizbi.

We took a few steps to verify our correlation. First, we devised an alternate classification based on certain unique characteristics of the "Message ID" SMTP header for Srizbi and MegaD bots, and verified that the classification does not change using that scheme. Second, we extracted last-hop relays from each classified message and checked overlaps between sets of botnet relays. The overlaps are small; it is never the case that many of the relays belonging to botnet X are also in the set of relays for botnet Y. The biggest overlap was 3.3% between Kraken and MegaD, which we interpret as 3.3% of Kraken's relays also being infected with the MegaD bot.

4.3.2 Spam campaigns

To gain insight into kinds of information spammers disseminate, we classified our incoming spam according to spam campaigns. We differentiate each campaign by the contents of the web pages pointed to by links in spam messages. Using data from Section 4.3.1, we classify our incoming spam according to botnets, and then break down each botnet's messages into campaign topics, defined by titles of campaign web pages. Table 3 shows these results for a single day of our trace. For example, Rustock participated in two campaigns – 22% of its messages advertised "Canadian Healthcare", while 78% advertised "MaxGain+". We could not fetch some links because of failed DNS resolution or inaccessible web servers; we marked these as "Unavailable". The table only shows the most prevalent campaigns; a group of less common campaigns is shown in row marked "Other".

All of our botnets simultaneously participate in multiple campaigns. For example, Kraken and Pushdo par-

[2]We exclude Grum from these analyses, because we have not yet monitored this recently discovered bot for a sufficiently long time.

	Kraken	MegaD	Pushdo	Rustock	Srizbi	Storm
Canadian Healthcare	0%	0%	0.01%	22%	3%	0%
Canadian Pharmacy	16%	28%	10%	0%	9%	6%
Diamond Watches	22%	0.1%	0%	0%	13%	0%
Downloadable Software	0%	0%	25%	0%	0%	0%
Freedom From Debt Forever!	19%	0%	0%	0%	0%	1%
Golden Gate Casino	0%	32%	0%	0%	0%	0%
KING REPLICA	0%	4%	3%	0%	15%	0%
LNHSolutions	0%	6%	0%	0%	0%	0%
MaxGain+ ... No.1 for PE	0%	0%	3%	78%	0%	0%
Prestige Replicas	7%	0%	0.3%	0%	31%	0%
VPXL - PE Made Easy	20%	8%	6%	0%	24%	55%
Unavailable	3%	22%	38%	0%	0%	24%
Other	13%	0.1%	15%	0%	5%	14%

Table 3: **Clustering incoming spam by the title of the web page pointed to by spam URLs.** The columns show how frequently each botnet was sending each campaign on April 30, 2008. Many botnets carry out multiple campaigns simultaneously.

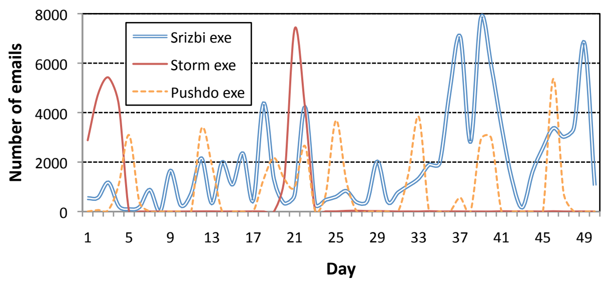

Figure 8: **Propagation campaigns.** The graph shows the number of e-mails with links that infected victims with either Srizbi, Storm, or Pushdo.

Botnet	Kraken	MegaD	Pushdo	Rustock	Srizbi	Storm
# unique relays seen	20,275	57,402	27,266	83,836	119,604	7,814

Table 4: **The number of unique relays belonging to each botnet.** These numbers provide a lower bound on the size of each botnet, as seen on September 3, 2008. More accurate estimates are possible by accounting for relays not seen in our spam feed.

ticipate in at least 5 and 7, respectively. The percentages give insight into how the botnet divides its bots across various campaigns. For example, Kraken might have four customers who each pay to use approximately 20% of the botnet to send spam for "Canadian Pharmacy", "Diamond Watches", "Freedom from Debt", and "VPXL". Multiple botnets often simultaneously participate in a single campaign, contrary to an assumption made by prior research [32]. For example, "Canadian Pharmacy" is distributed by Kraken, MegaD, Pushdo, Srizbi, and Storm. This suggests the most prominent spammers utilize the services of multiple botnets.

Botnets use different methods to assign their bots to campaigns. For example, Botlab monitors 20 variants of Srizbi, each using a distinct C&C server. Each C&C server manages a set of campaigns, but these sets often differ across C&C servers. For example, bots using C&C server X and Y might send out "Canadian Pharmacy" (with messages in different languages), whereas server Z divides bots across "Prestige Replicas" and "Diamond Watches". Thus, Srizbi bots are partitioned *statically* across 20 C&C servers, and then *dynamically* within each server. In contrast, all of our variants of Rustock contact the same C&C server, which dynamically schedules bots to cover each Rustock campaign with a certain fraction of the botnet's overall processing power, behaving much like a lottery scheduler [28].

4.3.3 Recruiting campaigns

Using our correlation tools, we were able to identify incoming spam messages containing links to executables infecting victims with the Storm, Pushdo, or Srizbi bot. Figure 8 shows this activity over our incoming spam trace. The peaks represent campaigns launched by botnets to recruit new victims. We have observed two such campaigns for Storm – one for March 13-16 and another centered on April 1, corresponding to Storm launching an April Fool's day campaign, which received wide coverage in the news [19]. Srizbi appears to have a steady ongoing recruiting campaign, with peaks around April 15-20, 2008. Pushdo infects its victims in bursts, with a new set of recruiting messages being sent out once a week.

We expected the spikes to translate to an increase in number of messages sent by either Srizbi, Storm, or Pushdo, but surprisingly this was not the case, as seen by matching Figures 8 and 7. This suggests that bot operators do not assign all available bots to send spam at maximum possible rates, but rather limit the overall spam volume sent out by the whole botnet.

4.3.4 Botnet membership lists and sizes

A botnet's power and prowess is frequently measured by the botnet's size, which we define as the number of active hosts under the botnet's control. A list of individual nodes comprising a botnet is also useful for notifying and disinfecting the victims. We next show how Botlab can be used to obtain information on both botnet size and membership.

As before, we classify our incoming spam into sourcing botnets and extract the last-hop relays from all successfully classified messages. After removing duplicates, these relay lists identify hosts belonging to each botnet. Table 4 shows the number of unique, classified relays for a particular day of our trace. [3] These numbers of relays for each botnet are effectively the lower bound on the botnet sizes. The actual botnet sizes are higher, since there are relays that did not send spam to University of Washington, and thus were not seen in our trace. We next estimate the percentage of total relays that we do see, and use it to better estimate botnet sizes.

Let us assume again that a bot sends spam to email addresses chosen at random. Further, let p be the probability with which a spam message with a randomly chosen email address is received by our spam monitoring system at University of Washington. If n is the number of messages that a bot sends out per day, then the probability that at least one of the messages generated by the bot is received by our spam monitors is $[1 - (1 - p)^n]$. For large values of n, such as when $n \sim 1/p$, the probability of seeing one of the bot's messages can be expressed as $[1 - e^{-np}]$.

We derive p using the number of spam messages received by our spam monitor and an estimate of the global number of spam messages. With our current setup, the former is approximately 2.4 million daily messages, while various sources estimate the latter at 100-120 billion messages (we use 110 billion) [12, 17, 25]. This gives $p = 2400000/110$ billion $= 2.2 \cdot 10^{-5}$.

For the next step, we will describe the logic using one of the botnets, Rustock, and later generalize to other botnets. From Section 4.1, we know that Rustock sends spam messages at a constant rate of $47.5K$ messages per day and that this rate is independent of the access link capacity of the host. Note that Rustock's sending rate translates to a modest rate of 1 spam message per two seconds, or about 0.35 KB/s given that the average Rustock message size is 700 bytes – a rate that can be supported by almost all end-hosts [13]. The probability that we see the IP of a Rustock spamming bot node in our spam monitors on a given day is $[1 - e^{-47500 \cdot 2.2 \cdot 10^{-5}}] = 0.65$. This implies that the 83,836 Rustock IPs we saw on September 3rd represent about 65% of all Rustock's relays; thus, the total number of active Rustock bots on that day was about $83,836/0.65 = 128,978$. Similarly, we estimate the active botnet size of Storm to be 16,750 [4].

These estimates rely on the fact that both Rustock and Storm send messages at a slow, constant rate that is unlikely to saturate most clients' bandwidth. Our other bots send spam at higher rates, with the bot adapting to the host's available bandwidth. Although this makes it more likely that a spamming relay is detected in our incoming spam, it is also more difficult to estimate the number of messages a given bot sends. In future work, we plan to study the rate adaptation behavior of these bots and combine it with known bandwidth profiles [13]. Meanwhile, Table 4 gives conservative size estimates.

[3]Since botnet membership is highly dynamic, we perform our calculations for a single day, where churn can be assumed to be negligible. As well, we assume DCHP does not affect the set of unique relays on a timescale of just a single day.

[4]These estimates conservatively assume a bot stays active 24 hours per day. Because some bots are powered off during the night, these botnet sizes are likely to be higher.

5 Applications enabled by Botlab

Botlab provides various kinds of real-time botnet information, which can be used by end hosts wishing for protection against botnets, or by ISPs and activists for law enforcement. Next, we discuss three applications we have built using our monitoring infrastructure.

5.1 Safer web browsing

Spam botnets propagate many harmful links, such as links to phishing sites or to web pages installing malware. For example, on September 24, 2008, we observed the Srizbi botnet distribute 40,270 distinct links to pages exploiting Flash to install the Srizbi bot. Although the current spam filtering tools are expected to filter out spam messages containing these links, we found that this is often not the case. For example, we have forwarded a representative sample of each of Srizbi's outgoing spam campaigns to a newly-created Gmail account controlled by us, where we have used Gmail's default spam filtering rules, and found that 79% of spam was *not* filtered out. Worse, Gmail filters are not "improving" quickly enough, as forwarding the same e-mails two days later resulted in only a 5% improvement in detection. Users are thus exposed to many messages containing dangerous links and social engineering traps enticing users to click on them.

Botlab can protect users from such attacks using its real-time database of malicious links seen in outgoing, botnet-generated spam. For example, we have developed a Firefox extension, which checks the links a user visits against this database before navigating to them. In this way, the extension easily prevented users from browsing to any of the malicious links Srizbi sent on September 24.

Some existing defenses use blacklisting to prevent browsers from following malicious links. We have checked two such blacklists, the Google Safe Browser API and the Malware Domain List, six days after the links were sent out, and found that *none* of the 40,270 links appeared on either list. These lists suffer from the same problem: they are *reactive*, as they rely on crawling and user reports to find malicious links *after* they are disseminated. These methods fail to quickly and exhaustively find "zero-day" botnet links, which point to malware hosted on recently compromised web servers, as well as malware hosted on individual bots via fast-flux DNS and a continuous flow of freshly-registered domain names. In contrast, Botlab can keep up with spam botnets because it uses *real-time blacklists*, which are updated with links at the instant they are disseminated by botnets.

5.2 Spam Filtering

Spam continuously pollutes email inboxes of many millions of Internet users. Most email users use spam filtering software such as SpamAssassin [24], which uses heuristics-based filters to determine whether a given message is spam. The filters usually have a threshold that a user varies to catch most spam while minimizing the number of false positives — legitimate email messages misclassified as spam. Often, this still leaves some spam sneaking through.

Botlab's real-time information can be used to build better spam filters. Specifically, using Botlab, we can determine whether a message is spam by checking it against the outgoing spam feeds for the botnets we monitor. This is a powerful mechanism: we simply rely on botnets themselves to tell us which messages are spam.

We implemented this idea in an extension for the Thunderbird email client. The extension checks messages arriving to the user's inbox against Botlab's live feeds using a simple, proof-of-concept correlation algorithm: an incoming message comes from a botnet if 1) there is an exact match on the set of URLs contained in the message body, or 2) if the message headers are in a format specific to that used by a particular botnet. For example, all of Srizbi's messages follow the same unique message ID and date format, distinct from all other legitimate and spam email. Although the second check is prone to future circumvention, this algorithm gives us an opportunity to pre-evaluate the potential of this technique. Recent work has proposed more robust algorithms, such as automatic regular-expression generation for spammed URLs in AutoRE [31], and we envision adopting these algorithms to use Botlab data to filter spam more effectively in real-time settings.

Although we have not yet thoroughly evaluated our extension, we performed a short experiment to estimate its effectiveness. One author used the extension for a week, and found that it reduced the amount of spam bypassing his departmental SpamAssassin filters by 156 messages, or 76%, while having a 0% false positive rate. Thus, we believe that Botlab can indeed significantly improve today's spam filtering tools.

5.3 Availability of Botlab Data

To make Botlab's data publicly available, we have set up a web page, `http://botlab.cs.washington.edu/`, which publishes data and statistics we obtained from Botlab. The information we provide currently includes activity reports for each spam botnet we monitor, ongoing scams, and a database of rogue links disseminated in spam. We also publish lists of current C&C addresses and members of individual botnets. We hope this information will further aid security researchers and activists in the continuing fight against the botnet threat.

6 Safety

We have implemented many safeguards to ensure that Botlab never harms remote hosts, networks, or users. In this section, we discuss the impact of these safeguards on the effectiveness of Botlab, and our concerns over the long-term viability of safely conducting bot research.

6.1 Impact on effectiveness

Initially, we hoped to construct a fully automatic platform that required no manual analysis on the part of an operator to find and analyze new botnet binaries. We quickly concluded this is infeasible to do safely, as human judgement and analysis is needed to determine whether previously uncharacterized traffic is safe to transmit.

Even with a human in the loop, safety concerns caused us to make choices that limit the effectiveness of our research. Our network sandbox mechanisms likely prevented some binaries from successfully communicating with C&C servers and activating, causing us to fail to recognize some binaries as spambots, and therefore to underestimate the diversity and extent of spamming botnets. Similarly, it is possible that some of our captive bot nodes periodically perform an end-to-end check of e-mail reachability, and that our spamhole blocking mechanism causes these nodes to disable themselves or behave differently than they would in the wild.

6.2 The long-term viability of safe botnet research

The only provably safe way for Botlab to execute untrusted code is to block all network traffic, but this would render Botlab ineffective. To date, our safeguards have let us analyze bot binaries while being confident that we have not caused harm. However, botnet trends and thought experiments have diminished our confidence that we can continue to conduct our research safely.

Botnets are trending towards the use of proprietary encrypted protocols to defeat analysis, polymorphism to evade detection, and automatically upgrading to new variants to incorporate new mechanisms. It is difficult to understand the impact of allowing an encrypted packet to be transmitted, or to ensure that traffic patterns that were previously benign do not become harmful after a binary evolves. Accordingly, the risk of letting any network traffic out of a captured bot node seems to be growing.

Simple thought experiments show that it is possible for an adversary to construct a bot binary for which there is no safe and effective Botlab sandboxing policy. As an extreme example, consider a hypothetical botnet whose C&C protocol consists of different attack packets. If a message is sent to an existing member of the botnet, the message will be intercepted and interpreted by the bot. However, if a message is sent to a non-botnet host, the message could exploit a vulnerability on that host. If such a protocol were adopted, Botlab would not be able to transmit any messages safely, since Botlab cannot know whether a destination IP address is an existing bot node. Other adversarial strategies are possible, such as embedding a time bomb within a bot node, or causing a bot node to send benign traffic that, when aggregated across thousands of nodes, results in a DDoS attack. Moreover, even transmitting a "benign" C&C message could cause other, non-Botlab bot nodes to transmit harmful traffic.

Given these concerns, we have disabled the crawling and network fingerprinting aspects of Botlab, and therefore are no longer analyzing or incorporating new binaries. As well, the only network traffic we are letting out of our existing seven botnet binaries are packets destined for the current, single C&C server IP address associated with each binary. Since Storm communicates with many peers over random ports, we have stopped analyzing Storm. Furthermore, once the C&C servers for the other botnets move, we will no longer allow outgoing network packets from their bot binaries. Consequently, the Botlab web site will no longer be updated with bots that we have to disable. It will, however, still provide access to all the data we have collected so far.

Our future research must therefore focus on deriving analysis techniques that do not require bot nodes to interact with Internet hosts, and determining if it is possible to construct additional safeguards that will sufficiently increase our confidence in the safety of transmitting specific packets. Unfortunately, our instinct is that a motivated adversary can make it impossible to conduct effective botnet research in a safe manner.

7 Related Work

Most related work can be classified into three categories: malware collection, botnet tracking systems, and spam measurement studies. We now discuss how our work relates to representative efforts in each of these categories.

Malware collection: Honeypots (such as Honeynet [11] and Potemkin [27]) have been a rich source of new malware samples. However, we found them less relevant for our work, as they failed to find any spam bots. The likely cause is that spam botnets have shifted to social-engineering-based propagation, relying less on service exploits and self-propagating worms. Other projects, such as HoneyMonkey [30], have used automated web crawling to discover and analyze malware; automated web patrol is now part of Google's infrastructure [20]. However, our results show that Google's database did not contain many malicious links seen in our outgoing spam feed, indicating that a blind crawl will not find malware from spam-distributed links.

Botnet tracking: Closely related to our work is the use of virtualized execution environments to track IRC botnets [22, 33]. By executing a large number of IRC bot samples, these efforts first identify the IRC servers and then infiltrate the corresponding IRC channels to snoop on the botnets. In our experience, botnets move away from plaintext IRC protocols to encrypted HTTP-based or p2p protocols, requiring more elaborate mechanisms as well as human involvement for a successful infiltration – a point of view that is increasingly voiced in the research community [14]. Less related to our work is the research on developing generic tools that can be deployed at the network layer to automatically detect the presence of bots [15]. Rishi [6] is a tool that detects the use of IRC commands and uncommon server ports in order to identify compromised hosts. BotSniffer [9] and BotHunter [8] are other network-based anomaly detection tools that work by simply sniffing on the network. Our work provides a different perspective on bot detection: a single large institution, such as University of Washington, can detect most of the spamming bots operating at a given point in time by simply examining its incoming spam feed and correlating it with the outgoing spam of known bots.

Spam measurement studies: Recently, a number of studies have examined incoming spam feeds to understand botnet behavior and the scam hosting infrastructure [1, 32, 31]. Botlab differs from these efforts in its use of *both* incoming and outgoing spam feeds. In addition to enabling application-level defenses that are proactive as opposed to reactive, our approach yields a more comprehensive view of spamming botnets that contradicts some assumptions and observations from prior work. For instance, a recent study [32] analyzes about 5 million messages and proposes novel clustering techniques to identify spam messages sent by the same botnet, but this is done under the assumption that each spam campaign is sourced by a single botnet; we observe the contrary to be true. Also, analysis of only the incoming spam feed might result in too fine-grained a view (at the level of short-term spam campaigns as in [31]) and cannot track the longitudinal behavior of botnets. Our work enables such analysis due to its use of live bots, and in that respect, we share some commonality with the recent study of live Storm bots and their spamming behavior [16].

8 Conclusion

In this work, we have described Botlab, a real-time botnet monitoring system. Botlab's key aspect is a multi-perspective design that combines a feed of incoming spam from the University of Washington with a feed of outgoing spam collected by running live bot binaries. By correlating these feeds, Botlab can perform a more comprehensive, accurate, and timely analysis of spam botnets.

We have used Botlab to discover and analyze today's most prominent spam botnets. We found that just six botnets are responsible for 79% of our university's spam. While domain names associated with the scams change frequently, the locations of C&C servers, web hosts, and even the content of web pages pointed to by scams remain static for long periods of time. A spam botnet typically engages in multiple spam campaigns simultaneously, and the same campaign is often purveyed by multiple botnets. We have also prototyped tools that use Botlab's real-time information to enable safer browsing and better spam filtering. Overall, we feel Botlab advances our understanding of botnets and enables promising research in anti-botnet defenses.

References

[1] D. S. Anderson, C. Fleizach, S. Savage, and G. M. Voelker. Spamscatter: Characterizing Internet Scam Hosting Infrastructure. In *USENIX Security*, 2007.

[2] Castle Cops. `http://www.castlecops.com`.

[3] K. Chiang and L. Lloyd. A Case Study of the Rustock Rootkit and Spam Bot. In *Proc. of the First Workshop on Hot Topics in Understanding Botnets*, 2007.

[4] R. Dingledine, N. Mathewson, and P. Syverson. Tor: the second-generation onion router. In *SSYM'04: Proceedings of the 13th conference on USENIX Security Symposium*, page 21, Berkeley, CA, USA, 2004. USENIX Association.

[5] D. Dittrich and S. Dietrich. Command and control structures in malware: From Handler/Agent to P2P. *USENIX ;login:*, 2007.

[6] J. Goebel and T. Holz. Rishi: identify bot contaminated hosts by IRC nickname evaluation. In *Proc. of the First Workshop on Hot Topics in Understanding Botnets*, 2007.

[7] J. B. Grizzard, V. Sharma, C. Nunnery, B. B. Kang, and D. Dagon. Peer-to-Peer Botnets: Overview and Case Study. In *Proc. of the First Workshop on Hot Topics in Understanding Botnets*, 2007.

[8] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. BotHunter: Detecting malware infection through ids-driven dialog correlation. In *USENIX Security*, August 2007.

[9] G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. In *NDSS*, 2008.

[10] T. Holz, C. Gorecki, K. Rieck, and F. Freiling. Measuring and Detecting Fast-Flux Service Networks. In *NDSS*, 2008.

[11] Honeynet Project. *Know your enemy*. Addison-Wesley Professional, 2004.

[12] Ironport. 2008 Internet Security Trends. `http://www.ironport.com/securitytrends/`, 2008.

[13] T. Isdal, M. Piatek, A. Krishnamurthy, and T. Anderson. Leveraging bittorrent for end host measurements. In *PAM*, 2007.

[14] C. Kanich, K. Levchenko, B. Enright, G. M. Voelker, and S. Savage. The Heisenbot Uncertainty Problem: Challenges in Separating Bots from Chaff. In *LEET*, 2008.

[15] A. Karasaridis, B. Rexroad, and D. Hoeflin. Wide-Scale Botnet Detection and Characterization. In *Proc. of the First Workshop on Hot Topics in Understanding Botnets*, 2007.

[16] C. Kreibich, C. Kanich, K. Levchenko, B. Enright, G. Voelker, V. Paxson, and S. Savage. On the Spam Campaign Trail. In *LEET*, 2008.

[17] Marshal Press Release: Srizbi now leads the spam pack. `http://www.marshal.com/trace/traceitem.asp?article=567`.

[18] MWCollect. `http://www.mwcollect.org`.

[19] J. Nazario. April Storms Day Campaign. `http://asert.arbornetworks.com/2008/03/april-storms-day-campaign/`, 2008.

[20] Niels Provos and Panayiotis Mavrommatis and Moheeb Rajab and Fabian Monrose. All Your iFrames Point to Us. In *USENIX Security Symposium*, 2008.

[21] Offensive Computing. `http://www.offensivecomputing.net`.

[22] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A Multifaceted Approach to Understanding the Botnet Phenomenon. In *IMC*, 2006.

[23] Shadow Server. `http://www.shadowserver.org`.

[24] SpamAssassin. `http://spamassassin.apache.org`.

[25] Spamunit. Spam Statistics. `http://spamunit.com/spam-statistics/`.

[26] S. Stover, D. Dittrich, J. Hernandez, and S. Dietrich. Analysis of the Storm and Nugache Trojans: P2P is here. *USENIX ;login:*, 2007.

[27] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. In *SOSP*, 2005.

[28] C. A. Waldspurger and W. E. Weihl. Lottery scheduling: Flexible proportional-share resource management. In *Operating Systems Design and Implementation*, pages 1–11, 1994.

[29] P. Wang, S. Sparks, and C. C. Zou. An Advanced Hybrid Peer-to-Peer Botnet. In *Proc. of the First Workshop on Hot Topics in Understanding Botnets*, 2007.

[30] Yi-Min Wang and Doug Beck and Xuxian Jiang and Roussi Roussev and Chad Verbowski and Shuo Chen and Sam King. Automated Web Patrol with Strider HoneyMonkeys. In *NDSS*, 2006.

[31] Yinglian Xie and Fang Yu and Kannan Achan and Rina Panigrahy and Geoff Hulten and Ivan Osipkov. Spamming Botnets: Signatures and Characteristics. In *SIGCOMM*, 2008.

[32] L. Zhuang, J. Dunagan, D. R. Simon, H. J. Wang, I. Osipkov, G. Hulten, and J. D. Tygar. Characterizing Botnets from Email Spam Records. In *Proc. of Workshop on Large-scale Exploits and Emergent Threats*, 2008.

[33] J. Zhuge, T. Holz, X. Han, J. Guo, and W. Zou. Characterizing the irc-based botnet phenonmenon. Technical Report TR-2007-010, Reihe Informatik, 2007.