

Evaluating Email’s Feasibility for Botnet Command and Control

Kapil Singh Abhinav Srivastava Jonathon Giffin Wenke Lee

School of Computer Science, Georgia Institute of Technology
{ksingh, abhinav, giffin, wenke}@cc.gatech.edu

Abstract

The usefulness of email has been tempered by its role in the widespread distribution of spam and malicious content. Security solutions have focused on filtering out malicious payloads and weblinks from email; the potential dangers of email go past these boundaries: harmless-looking emails can carry dangerous, hidden botnet content. In this paper, we evaluate the suitability of email communication for botnet command and control. What makes email-based botnets interesting is the lack of clear detection and mitigation strategies that defenders could use to disrupt the botnet. We first demonstrate that botnet commands can remain hidden in spam due to its enormous volume. If email providers deploy specialized detection of spam-based botnets, botmasters can alternatively communicate with bots via non-spam email that cannot be safely discarded. We show the viability of such communication by means of simulations and a prototype, and we discuss the limited prospects for detection of email botnets.

1. Introduction

Internet threats have recently transformed from highly visible, disruptive attacks to stealthy attacks used for profit. At the center of this change are the networks of compromised machines running bots under the control of botmasters. These botnets are the workhorses of many online criminal enterprises—bots send spam, perform DDoS attacks for extortion, or host phishing web sites. The botmaster controls each bot in a botnet via a command and control (C&C) channel that connects the botmaster with each bot. A successful botnet requires a stealthy or hidden C&C channel. If defenders identify a channel, they can disrupt the communication between the botmaster and his bots, and the botnet is no longer effective.

This paper analyzes email’s ability to provide a stealthy C&C channel. Email communication is already problematic: message delivery is occasionally unreliable, spam is a nuisance to users, and malicious messages may phish personal data from users or include unsafe executable software as an attachment. We show that email and email spam threaten system and network security beyond the dangers already known—they provide feasible and hidden communication channels suitable for botnet control. Botnet com-

mands embedded in messages adversely affect the operation of the Internet beyond the bandwidth and storage consumed by the spam. Evidence indicates that malicious individuals are already passing covert information via email and web page spam [14].

Our specially crafted messages implement a botnet C&C mechanism that succeeds with no user interaction at an email client and remains stealthy against current botnet detection mechanisms. Bots automatically extract hidden commands from email messages anytime the victim machine has network connectivity. Our channels remain stealthy because identification of messages containing bot commands requires significant computation time per message, even if the defender knows all information about the communicating bot. For an email provider such as Gmail, this identification time totals thousands of years of compute time per day.

We evaluate a botmaster’s ability to encode commands in email by designing two C&C channels. Our first, spam-based channel leverages useful characteristics of spam:

- *Spam has become normal.* People expect to receive junk email, and email providers expect to identify spam messages sent to their users.
- *Email providers automatically filter spam out of inboxes.* Users tend to ignore spam that bypasses their inbox, and spam messages hence often persist in an account until the email provider deletes the messages after a timeout (e.g. 30 days).
- *Spam detection is imperfect.* Email providers do not automatically delete all messages that trip their spam detection algorithms due to the occasional legitimate email flagged as spam. Provided that our messages do not hit spam blacklists, we can reasonably expect the spam to reach the target account.
- *Spam messages have weak structure and meaning,* providing flexibility to a botmaster embedding botnet commands into the spam.
- *The enormous volume of spam* eases a botmaster’s ability to hide messages containing commands.

Our bots may generate anomalous access patterns to an email account’s spam folder. Email providers currently do not regulate access to spam, though we suggest that restricting access to spam messages may be an effective future strategy to combat spam-based C&C channels. Unfor-

tunately, the principles of our spam-based channel likewise apply to non-spam email. Although it is more difficult for a botmaster to generate non-spam email than spam, we show with our second C&C design that hidden bot commands can be successfully distributed via non-spam messages.

We demonstrate the operational viability of email-based C&C by means of a prototype. We successfully encoded commands in both spam messages that Gmail automatically filtered to a spam folder and in unfiltered non-spam messages, and our bot successfully extracted the commands from the emails. We show that it is computationally impractical for defenders to detect the email-based C&C channel, *even if they know the complete code, data, and behavior of the bot*. The botmaster drives this computation time, and he can arbitrarily increase the time to counter defenses based on increased hardware commitment. This claim is supported by our simulation results, which demonstrate that by controlling certain parameters, the botmaster can achieve reasonable tradeoffs between botnet response time and stealthiness in real-world deployments.

By anticipating the (malicious) innovations that may occur among bot developers, defenders are better able to limit the effects of botnets. In this case, improving the safety of email communication requires strategies designed to detect the presence of hidden commands, or to prevent bots from retrieving messages with those commands. Although the defensive prospects seem limited, we conclude with suggested alterations to email clients and servers that may constrain botmasters attempting email-based C&C.

2. Related Work

Many current botnets use a centralized IRC channel for command and control [6]. Centralized C&C suffers from two critical shortcomings: first, a botmaster loses control of his botnet once the C&C servers are deactivated by defenders, and second, the entire botnet is exposed once a C&C server is hijacked or captured [11]. Active monitoring of public IRC servers can also reveal malicious rallying or suspicious messages [9, 17]. In contrast, our email C&C substrate produces robust botnets. Although central servers store email, the servers cannot be deactivated without widespread disruption of legitimate email service.

Online information sources, such as newsgroups, blogs, or websites located using a search engine, provide C&C channels that seem unlikely to raise suspicion [20]. However, the servers storing commands are centralized and can be disabled or simply dropped from search engine results. Newsgroups and blogs suffer from the “many eyes” problem [18]: fake or suspicious blogs are often reported by readers and consequently deleted. A defender who acquires and analyzes a bot can extract the centralized C&C source from its code, allowing subsequent disabling or monitoring of the channel. In contrast, our email-based channels mix encoded messages with normal email, requir-

ing defenders to perform a computationally costly analysis of large volumes of email.

Peer-to-peer (P2P) communication [3, 4, 12] offers advantages not found in centralized C&C. Even if defenders identify a subset of the bots in a botnet, communication among the remaining bots will not be disrupted. However, botmasters developing P2P C&C channels have preferred simple designs [3, 4] resulting in botnets that are immature and prone to detection. For example, Phatbot uses Gnutella cache servers to retrieve the list of bots in the botnet [3], making it susceptible to detection. Sinit finds other bots via easily identified random probing [4]. By using email as our C&C channel, our design neither needs complex protocols nor generates anomalous network traffic. P2P botnets also suffer from the problem of losing bots whenever those bots change their dynamic IP addresses. Email channels are resilient to changing IPs.

A bot running on a machine must find a path to the Internet. Botnets communicating via IRC or P2P face difficulties due to common local network architectures. When running behind a firewall or NAT device, the bot may be unable to open ports at the firewall or forward ports from the NAT device to the machine running the bot. Our email-based C&C channel easily handles such complexities. The bot initiates network communication with common public mail service providers, so it easily traverses firewalls and NAT devices.

Email communication has one additional, attractive property that benefits botnet creators. Unlike other centralized approaches, there is no fixed endpoint from which the botmaster must transmit commands. The botmaster can send encoded messages from any machine and any source email address, and he can frequently change these source addresses to evade detection by law enforcement agencies.

Nazario et al. [16] have discussed the idea of using spam as a stealthy covert channel to receive worm updates. They did not analyze the channel’s feasibility or provide a prototype implementation for experimentation. They discussed the possibility of spam-based communication in centralized Usenet and mailing lists, which again suffer from most problems faced by IRC. Kret [14] presented a similar idea using a case study of an existing IRC group that uses spam as a covert communication channel. These examples clearly attest to the possibility of threats presented in this paper. We provide a more generic framework for botnet communication both using spam as well as non-spam messages. To the best of our knowledge, we are the first to exploit the email medium for botnet C&C.

3. Proposed Email-Based Botnet

Our goal is to evaluate the suitability of email for botnet command and control. A botmaster must be able to transmit commands to bots installed on victim computers in a way that cannot be easily disrupted by defenders or

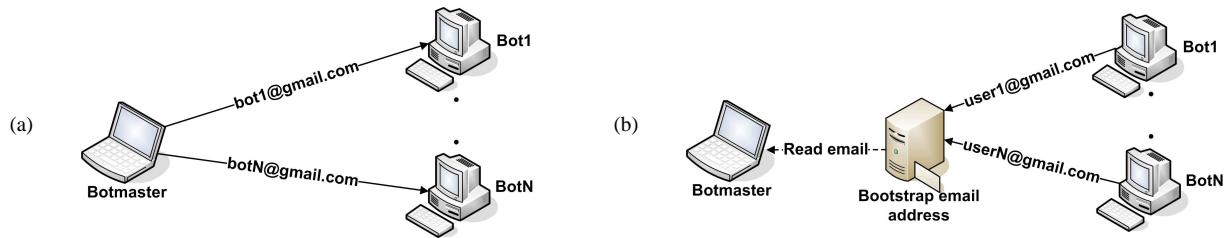


Figure 1: Construction Process: (a) Method 1: Botmaster creates bot identities. (b) Method 2: Bot learns the user identity and passes it to the botmaster.

law enforcement personnel. We assume that bots have already been installed on a collection of victim machines. Our work then provides subsequent communication from the botmaster to each bot over an email communication channel that cannot be easily identified or blocked.

We expect that powerful defenses exist to detect and disrupt botnets. In our threat model, we assume that defenders have access to a bot binary and are able to learn the bot’s entire execution behavior through forensic analysis techniques. This allows defenders to identify the command list and the algorithms used by the bot to extract commands from email messages. Simply stated: the threat model allows defenders to know everything that is known to the bot. We architected our system so that a defender who captures and analyzes a bot cannot leverage knowledge of that bot’s behavior to disrupt other botnet communication.

Like other architectures, our email-based botnet operates in two phases. First, bots use a centralized point to bootstrap future communication. In this construction phase, each bot acquires and registers an email address with the botmaster. In the subsequent decentralized operational phase, the botmaster sends email messages to the previously registered email addresses. The first construction phase is described in Section 4. Section 5 presents two email blending strategies—spam-based and non-spam-based—of the operational phase.

4. Botnet Construction

In the initial construction phase, the botmaster learns the identity of each bot in his control. The identities provide the botmaster with the ability to communicate with the bots; in an email-based botnet, email addresses act as bot identifiers. A bot can acquire its identity in two ways. First, the botmaster can assign each bot an email address and pass this address to the bots during recruitment. Second, the bot can choose its own address by either acquiring a new email account itself or by hacking into an account used by the victim machine’s user, and it can communicate that address back to the botmaster. The repository of bot email addresses is maintained solely by the botmaster, and hence others cannot hijack the botnet’s control.

In the first case (Figure 1(a)), the botmaster creates an email address for each bot in his network. The botmaster is

essentially choosing the identity of each bot and hence all botnet communication is one-sided (from the botmaster to the bots). This approach is practical only when the process of acquiring email accounts can be automated. Many email providers require humans to identify a visual CAPTCHA image [21], and it would be cumbersome for the botmaster to manually create these email accounts. However, known workarounds [15] may allow the the botmaster to create software that automatically passes such tests.

There are limitations to the creation of bot-specific accounts. First, the creation of a number of accounts that do not send or receive legitimate emails could be an anomaly detected by email providers. Second, the email providers could also develop better visual tests that cannot be broken by the algorithms that defeat current CAPTCHA tests.

The botmaster may hence prefer to use existing legitimate email accounts for botnet communication. Each bot can sniff the username and password of the user of the victim machine using some mechanism not visible to the user. For instance, our prototype design used a simple keylogger to learn these user details. Many malware carry keyloggers as part of their arsenal [13] and are already leaking private user information to the malware authors.

After a newly installed bot acquires access to an email account, it then sends its identity (the email address) to an email address that can be easily monitored by the botmaster (Figure 1(b)). This centralized address is specified in the bot binary and can be one or multiple email addresses. If defenders using forensic analysis techniques can quickly identify the rallying address, they could shut down the email account or monitor messages delivered to the account. To address this defense, the botmaster may use one of the following procedures:

- Create the bootstrapping email account at an email provider that prevents defenders from shutting down or monitoring the address. For example, the provider can be in a country with no cyber laws or international collaboration.
- Use several email addresses for bootstrapping instead of a single email address.
- Delete the rallying email address immediately upon receiving the bots’ identities.

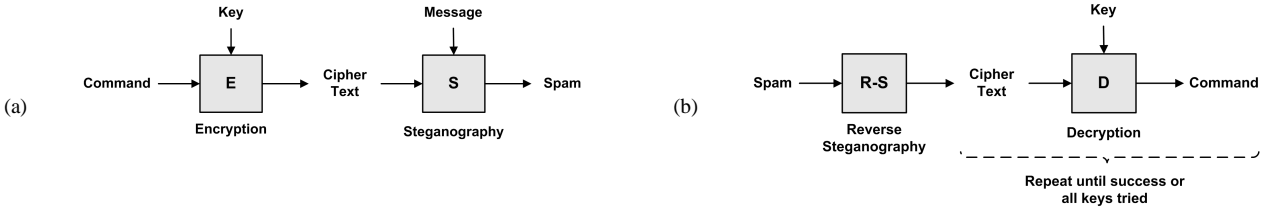


Figure 2: Spam Blending: (a) Botmaster blends the command with the spam message. (b) Bot extracts the command from spam.

- Run an email service at one of the bots, and scrub the bot from the machine after receiving all identity information. Defenders face difficulties monitoring or disabling the account because the botmaster does not use a public email provider.

The bots may need to provide periodic feedback information to the botmaster. They can communicate with the botmaster via a frequently changing email address given to bots as part of a command from the botmaster. This feedback design is similar to the monitoring mechanism used by P2P botnets [22].

Bots may choose from a wide collection of suitable types of email service, including web-based service, IMAP, and POP3. Some web-based email providers prohibit access to spam folders through any client interface other than a web browser. However, free tools providing a POP3 proxy to popular web-based services [1] such as Gmail and Yahoo! mail are publically available. These tools provide the flexibility to download any folder from the email account without marking messages as read, and a bot can simply use any such tool to access emails via POP3.

5. Email Blending Strategies

We have developed two email blending strategies: commands blended with spam (Section 5.1) and commands blended with non-spam email (Section 5.2). Spam blending is the simpler technique. It is easier to know that a message is likely to be marked as spam than it is to know that a message will not be marked. As a result, spam blending is the likely command channel until such time that enough botmasters employ spam blending to gain the notice of email service providers. The email providers may then use anomaly detection or spam access restrictions to limit the propagation of bot commands in spam. At that time, botmasters can fall back to the more difficult non-spam blending approach.

5.1. Spam Blending

Once botnet construction is complete, the botmaster has an operational communications channel to each bot in his botnet. Each bot periodically checks its emails for messages containing commands from the botmaster. A naïve way to send commands is to pass them in cleartext as part of an email. However, email providers can efficiently

search for known commands as part of their spam processing, and they can then drop all messages containing commands. A better approach hides the command in email so that network-based botnet detection mechanisms must expend significant computation time to identify the presence of the C&C channel.

Hiding the command in spam rather than in normal email provides an easy way for commands to persist in the email account, giving the bot time to extract the commands. The botmaster *wants* his messages to be detected as spam. If the service provider filters the message to the spam folder, then the command will persist until either the user chooses to inspect his spam or some spam expiration date is reached. This gives the bot an increased opportunity to retrieve the command from spam emails.

Our first C&C design takes this approach: we blend bot commands with spam to produce a new spam message containing the hidden command. Figure 2 shows the blending mechanism used by our prototype botnet. The botmaster blends commands with spam using a combination of encryption and steganography, and bots later reverse these operations to extract the command from the spam message.

Hiding commands in spam using steganography makes it difficult for humans or email providers to identify the presence of a command within a spam message. It is straightforward to hide content within a spam due to the unstructured content already present in spam messages. A human reading the message will see just another spam with unstructured content. There is no noticeable difference between our encoded spam and other spam messages that contain no dangerous content. Furthermore, without knowing the particular steganographic algorithm used by the botmaster, an email provider is unlikely to identify the presence of a command using automated means.

However, the provider could learn the steganographic algorithm via analysis of the reversal function evident in a captured bot's code. Hence, we encrypt the botnet commands to complicate an email provider's ability to determine the presence of a command. They could run the steganographic reversal algorithm on a spam message, but it would not return data that was a clear bot command. Only by additionally reversing the encryption can the provider realize that the data actually was an encrypted command. Carefully chosen encryption algorithms make it computationally prohibitive for the defenders to decrypt

```
Hi,

Via_aagra $3. 35
Va_aalium $1. 25
Cia_aalis $3. 75
Xan_nnax
Som_mma

http://www. kedrx .com

Remove spaces in the above link

Muscle sir no space semiopened Dobby i slipshod mother. Mond sir sign
sing pigskin no. Small-footed no tried lamia placarder part. Taen
unsevereness phylarchic tfc and poisoning stand sir. Prettify time-served
talpid sportswriters and undry song space. Skaffie pre-islamite mesaraic
napkins and linguanasal thought part.
```

Figure 3: A spam message with the embedded encrypted botnet command “.advscan lsass 200 5 0 -r -s”.

the messages *even if they possess a captured bot*, as we discuss in Section 7.

A bot or email provider inspecting a spam message reverses the steganography and encryption to extract the actual command. A bot processes only the small number of spam messages received by an individual email account. An email provider hoping to identify spam messages comprising a C&C channel must analyze the entire volume of spam messages traversing their servers.

Spam blending works surprisingly well. Figure 3 shows a spam message blended with a standard botnet command. Spam messages typically contain some irrelevant words to prevent Bayesian filtering of the message. The blended spam message looks very similar to a normal spam message with random sequences at the end. This shows the effectiveness of the steganographic algorithms and suggests that straightforward detection of emails in the C&C channel may be difficult. To both a human and an email provider, it would look like any other spam message and would be ignored as harmless.

Encoding commands into spam messages has limitations that could be used by defenders to identify the botnet commands. While many people receive spam, it seems likely that almost no one actually reads it. Thus, a bot may access its email account with unusual access patterns. If some client attempts to read all spam, the email provider could flag this account as anomalous. Bots could escape such detectors by lowering their spam harvest rates below the anomaly threshold used by the provider. The bot could read only a fraction of the spam to appear human, and the botmaster can send multiple messages with the same encoded command to each bot to maximize the probability that the bot processes at least one blended spam.

If spam-based C&C channels become widespread, email providers may employ countermeasures that restrict access to spam. For example, they could require users to pass a visual test (CAPTCHA) before gaining access to the contents of their spam folder. The email interface can also include last spam reading time so that users could

see that their spam is being read without their knowledge. Finally, the computational requirements imposed on the email providers to detect a channel could be lessened if the providers only apply the computation to spam messages that clients actually bothered to read.

5.2. Non-Spam Email Blending

To combat countermeasures against spam-based channels, botmasters could instead blend commands with unfiltered non-spam email. Accessing emails stored in an inbox is a normal user behavior; hiding the command in normal email allows the bot to successfully receive commands while mimicking a human user’s email usage behavior. Therefore, it does not present an anomaly that can be used by email providers for detecting the botnet C&C channel. In this blending strategy, the botmaster wants to prevent the email from being flagged as spam.

The botmaster’s message generation difficulty is much less than the difficulty faced by actual spammers hoping to avoid detection. Anti-spam mechanisms will be largely ineffective against email containing embedded C&C traffic because the botmaster’s messages need not show the same inherent properties as spam. The can be made to look like normal messages coming from friends or family. As a result, email providers cannot prevent such C&C messages from reaching each inbox.

Email providers could employ aggressive, per-client spam detection for identifying command-carrying non-spam messages. Bayesian filters [7] learn a user’s expectation of normal email and filter messages differing from that expectation as spam and have been shown to exhibit high accuracy [19]. However, aggressive filtering suffers from false positives: legitimate email may be erroneously marked as spam. Subsequently, the email provider must allow clients to pursue their spam folders to manually reclassify false positives as normal. The end result: email providers either filter spam aggressively but allow easy spam access, and our spam blending design is operationally useful, or they regulate access to spam folders but filter cautiously, and our non-spam blending design succeeds.

Moreover, the bot can pass information, such as sample emails, from the user account during the construction phase. This can be used by the botmaster to extract user’s email patterns and to create blended emails closely corresponding to the messages already received by the user. This improves the chances of the emails to pass through the spam filters even when the filtering level is high.

The non-spam blending approach increases the visibility of the command-carrying message, and the human user may delete the message before the bot has received the command. The botmaster can increase the probability of bot receiving the command by reducing the interval between subsequent email checking by the bot or by sending multiple message to the bot over a period of time. The bot can also read the message at the same time as the user.

	Non-spam	Spam
Gmail	100.0%	99.8%
Yahoo!	100.0%	94.8%
SpamAssassin	100.0%	94.2%

Table 1: Percentage of correct classification of automatically generated non-spam and spam messages containing an embedded encrypted bot command. We sent 1000 messages of each type to each of three different spam filters.

6. Keying Strategies

The botmaster can choose the particular encryption and steganographic algorithms used. For encryption, we suggest two keying strategies that each provides the foundation for stealthy C&C.

In the first keying strategy, each bot uses a different decryption key. The keys can be generated by the botmaster and passed to the bots (Figure 1(a)), or the bots can create their own keys and communicate them to the botmaster during construction (Figure 1(b)). Even if the defenders are successful in learning the complete bot behavior in accordance with our threat model, they would only learn the keys of the captured bots. Hence, the botmaster can still successfully command the bots whose keys are not known to the defenders.

This approach has a high tolerance against detection, but it increases the management overhead for the botmaster. The botmaster has to keep a mapping of the different keys corresponding to different bots and also must generate a different spam message for each bot. From our experience, we believe that botmasters prefer simpler C&C approaches that require only small effort and management. Hence, some botmasters may not prefer this keying strategy due to its per-bot management overhead.

In our second keying strategy, no bot has any decryption key. After running the reverse steganographic algorithm on a newly received email, each bot must determine if the recovered data is an encrypted command and the contents of the actual command. The bot has knowledge of the particular encryption algorithm used by the botmaster, but has no knowledge of the key. The bot recovers commands via exhaustive testing of keys until some key decrypts the data into a legitimate command. For practicality, we set an upper bound on the key space through which the bot will test keys. If the bot exhausts the key space and never decrypted the data into a command, then the spam was not part of the C&C channel. An analyst inspecting a captured bot will recover no key material at all, because the bot contains no key material.

This strategy eliminates the management overhead of the previous strategy. The botmaster may send the identical message, and hence identically-encrypted commands,

Δ	Time interval between subsequent email checks of the bot
m	Number of messages received by a user i per time Δ
M	Number of messages received by a user i per day
N	Total number of users for an email provider
t	Time to process a message looking for bot commands
C	Ratio of the processing capacity of the email provider compared to a typical user machine
u	CPU usage of the bot ($0 \leq u \leq 1$)
l_k	Key length
l_m	Message length

Table 2: Variables used in the Section 7 analysis.

to all bots in his network. The botmaster has the freedom to rekey at will, so defenses that retry previously identified keys will not be successful.

A potential shortcoming occurs due to the brute-force decryption. It is possible that reverse steganography of an unblended email message may produce data that will decrypt to a command for some key in the key space. This is a false positive: a bot extracts a command in violation of the botmaster’s intent. We do not presently know if such false positives would occur in practice.

7. Experiments and Analysis

We studied the feasibility of email-based botnet communication and the stealthy nature of the channel. Section 7.1 describes tests using a prototype bot and a collection of spam and non-spam email. Section 7.2 compares the computational overhead of command extraction via exhaustive key search from an email message at a single bot with the overhead incurred by an email provider. An email-based communication channel does have greater latency than previous C&C channels, and Section 7.3 analyzes our latency. Section 7.4 uses a botnet simulation to show how we expect the email channel to affect latency in actual deployments.

7.1. Prototype Bot

We developed a prototype bot that demonstrates the feasibility of email-based botnet C&C. The bot consists of a simple email client to send and receive mails from an email account and a stealthy keylogger to learn the user’s email account and password. For our prototype, we used KGB as the keylogger and FreePOPs [1] as the email client offering POP3 access to Gmail’s web interface. We encrypted a bot command using DES and generated a spam message embedded with the encrypted command using a steganographic tool called NiceText [8]. Gmail still flagged our specially crafted message (Figure 3) as spam and automatically filtered it to the spam folder. The bot successfully retrieved this spam message and was able to extract the hidden command. Our bot also has the capability to receive botnet commands in non-spam messages. It was able to successfully extract the botnet command from our crafted

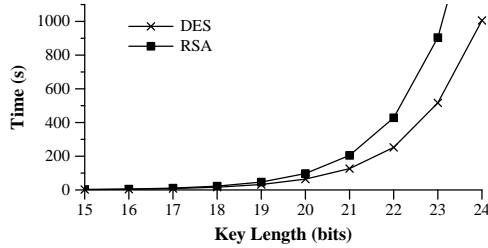


Figure 4: Time to decrypt messages via brute-force key search, for various maximum key lengths. The plaintext was 30 bytes long.

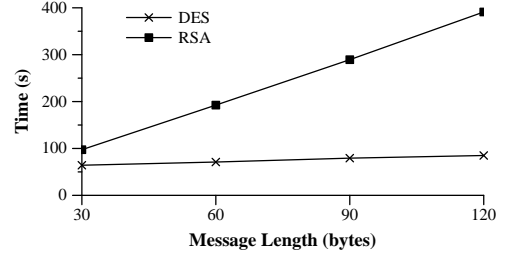


Figure 5: Time to decrypt messages via brute-force key search, for various plaintext message lengths. The key contained 20 unknown bits.

email message that was received in Gmail’s inbox.

We studied the feasibility of creating command carrying messages by generating a set of such messages and testing them against current spam filters. We ran a set of experiments for both spam and non-spam messages; our goal was to show that a botmaster could use small source data to easily and automatically generate large volumes of both spam and non-spam email carrying a hidden bot command. We trained the NiceText tool to build two separate dictionaries corresponding to the two message categories. For spam messages, we obtained a small training set of just 10 spam messages from a public spam database. For non-spam, we acquired 10 emails from the Linux mailing list LWN [2]. After learning the structure of existing messages, NiceText generated a grammar for each class of messages. It used these grammars to create spam and non-spam email messages embedded with an encrypted bot command which we then sent to Gmail, Yahoo!, and a locally-installed SpamAssassin filter. We initially discovered that NiceText insufficiently added spam keywords to spam messages, so we augmented NiceText with an automated post-pass component that inserted advertising links and keywords into spam in a manner known to the bots. Table 1 shows the results of our experiment. Gmail correctly filtered 99.8% of the spam messages into the receiver’s spam folder, Yahoo! filtered 94.8%, and SpamAssassin filtered 94.2%. All three spam filters had 100% accuracy for non-spam messages as they delivered all messages to the recipient’s inbox.

7.2. Overhead Analysis

Our threat model assumes that the common decryption algorithm used by the bots is also known to the defenders. The defenders can pass the algorithm to email providers and they, in turn, can use it to extract the commands from messages. The email provider then can drop such messages and alert the recipient about the possible attack.

Even if the defenders know the decryption algorithm used by the bot to extract commands, it is impractical to actually do so. The issue is *scale*: a bot must identify and extract commands from the small number of messages received each day by a single email account. An email provider hoping to identify messages comprising C&C

channels must search all messages sent to all its users. The botmaster has control over what encryption algorithm and keying strategy is used by the botnet, and he can intentionally make the algorithm computationally expensive. This expense can be easily borne by a bot processing a small number of messages, but it grows prohibitively costly for a large email provider. The number of messages processed by an email provider such as Gmail or Yahoo! is many orders of magnitude larger than the number of messages processed by a bot.

The botmaster has full control over the amount of computation necessary to extract commands from an email message. He can adjust three parameters—maximum key size for exhaustive search l_k , message length l_m , and the encryption algorithm used—giving him tremendous flexibility when creating a stealthy C&C channel. Increasing the maximum key size or padding the message to increase its length increases the time necessary for brute-force decryption. He can choose these values based on the delay he deems acceptable for bots to extract commands. The choice of encryption algorithm also affects cost—for example, RSA is computationally more expensive than DES. Figures 4 and 5 show the time necessary to decrypt messages via exhaustive key search for two different algorithms, a variety of key sizes, and several message lengths. We computed these times with a Java program running on a 2.40 GHz Pentium 4 machine running Linux; a botmaster building these curves should execute tests at native speed. We produced a key sized appropriately for the encryption algorithm by padding a shorter key with random bits known to both the botmaster and the bot.

For a bot, the overhead to process a day’s worth of email looking for commands is given by:

$$T_{bot} = \frac{Mt}{u} = \frac{Mf(l_k, l_m)}{u} \quad (1)$$

where variables are given in Table 2. The function $f(l_k, l_m)$ characterizes the relationship among message extraction time, key length, and message length. It is of order $O(l_m 2^{l_k})$ determined by the graphs in Figures 4 and 5. Note that the overhead is for receiving commands. This does not include the overhead for using the bot for a par-

ticular purpose (such as password cracking); that overhead would be similar for any type of botnet.

When the processing capacity of the email provider is used at full capacity ($u = 1$), the corresponding overhead at the email provider is given by:

$$T_{ESP} = \frac{(\sum_{i=1}^N M_i) f(l_k, l_m)}{C} \quad (2)$$

The total number of messages processed by an email provider is orders of magnitudes greater than the messages received by a user, i.e., $\sum_{i=1}^N M_i \gg M_j$ for any j . Therefore, we can see that $T_{ESP} \gg T_{bot}$.

Consider the concrete case of exhaustive key search at an email provider searching for a spam-based C&C channel. Using conservative estimates, we suppose that a typical email user receives 25 spam messages per day. These messages do not trigger automatic drop rules based on blacklists, although they are flagged as spam by the email provider and automatically filtered to the user's spam folder. There are about 250 million worldwide users of Yahoo! mail [5]. Supposing that the botmaster is using DES with a maximum key length of 18, then the time to process each spam message is about 15 seconds (Figure 4). The total overhead for Yahoo! to process *one day's worth of spam* directed to its users is then $(25) \times (250 \text{ million}) \times (15 \text{ seconds}) \approx 2,970$ years. Gmail, with 51 million users [5], would take about 606 years to process a single day's spam. Note that processing times may be slightly lower in practice because duplicate spam messages need only be analyzed once by the email provider. However, the random blocks of text inserted by many spammers at the end of messages complicate a provider's ability to identify duplicate messages.

An email provider could address scale with additional processing power (i.e. increase C). The ratio between overhead at an email provider and overhead at a single bot is:

$$\frac{T_{ESP}}{T_{bot}} = \frac{(\sum_{i=1}^N M_i)/M}{C/u} \quad (3)$$

If the ratio C/u is close to the ratio $(\sum_{i=1}^N M_i)/M$, then the overall ratio between overhead at the provider and at each bot would be close to 1. Again, we can consider concrete values. For Yahoo! and an average user, the numerator is 250 million. If the botmaster designs his bots to use 2% of each victim's CPU ($u = 0.02$), then Yahoo! can keep pace with bots decrypting email with computing power 5 million times stronger than an average user ($C = 5$ million). Although Yahoo! could feasibly deploy 5 million machines, we doubt that any company would be willing to commit such resources towards botnet C&C detection.

7.3. Latency Analysis

The latency for a botnet C&C is the time between the botmaster's issue of a command and a bot's receipt of the

command. We expect our email-based channels to have greater latency than IRC and P2P channels, largely because our bots only periodically check their email account for new messages. Ignoring negligible network delay, the maximum latency L for an email channel is the combination of the wait time before accessing new mail and the time required to process the new messages received.

$$L = \Delta + \frac{m\Delta f(l_k, l_m)}{u} \quad (4)$$

Latency depends on parameters m , Δ , l_k , l_m and u . While the value of m is specific to a user's account, other parameters can be controlled by the botmaster to strike a balance between latency and stealthiness: lower latency for the bot maps to lower processing requirements at the email provider and hence poorer stealth for the botnet C&C.

The number of messages m received by the bot per time Δ is an important consideration. If m is large, then the email processing time would be the dominating factor in latency; otherwise, Δ would dominate. If the volume of email is too heavy, then the bot cannot keep up with the rate of incoming messages. Therefore, the values of l_k and l_m should be fine tuned for every bot or chosen once so that most bots will successfully keep pace with the rate of incoming email.

7.4. Simulation Results

The previous section mathematically analyzed the expected latency of an email C&C channel. In this section, we carry out a series of simulations to demonstrate the viability of a typical email-based botnet in practice. We show that the channel provides the botmaster with reasonable tradeoffs in botnet response time and stealthiness.

We define *Guarantee of Support (GoS)* as the percentage of bots responding within a predefined *desired time*. The desired time is the maximum latency acceptable to the botmaster for a successful botnet attack. This time value may vary according to the functionality of the botnet. For example, it is likely acceptable to have a higher response time for botnets used to generate spam. Conversely, the botmaster likely wants lower latency in botnets used for DDoS attacks or distributed password cracking.

We simulated 10,000 bots all processing email messages for the botnet commands. We used probabilistic distributions for parameters that may differ at different bots. More specifically, we model the rate at which each victim receives new mail, their time to process each message, and the delay between successive checks for new mail. We used a Poisson distribution to model the number of new messages received per day. The mean for this distribution was taken as 300 for the experiments in Figures 6b and 6c. We modeled the time required to process a single message by a Gaussian distribution. We used the processing time measured experimentally on a common, one-year-old desktop machine (Section 7.2) as the parameters of the distribution:

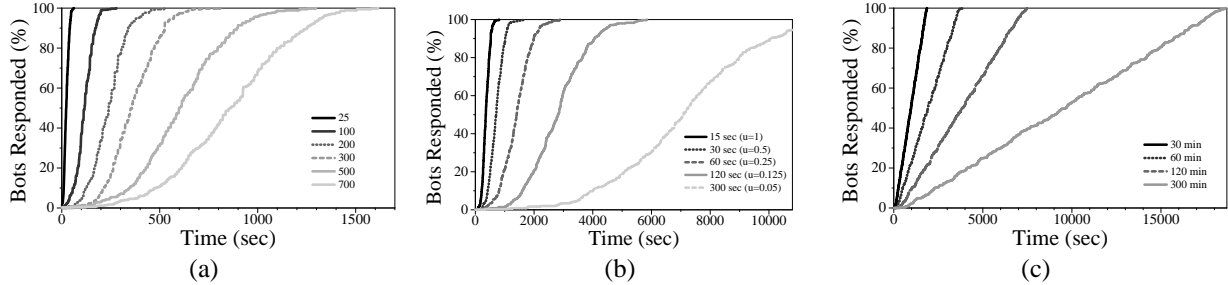


Figure 6: Simulation results: Effect of (a) count of emails received per day [variable M], (b) bot computation time per message [variable t], and (c) time between successive checks for new mail [variable Δ] upon the Guarantee of Service. In (a), values in the legend are the means for a Poisson distribution over different user accounts. In (b), values in the legend are means for a Gaussian distribution with variance 33% of the mean. In (c), values in the legend are absolute values of Δ .

mean 15 seconds, variance 5 seconds (Figures 6a and 6c). Finally, we fixed the time between checks for new mail at 120 minutes for the simulations in Figures 6a and 6b. Figure 6 presents the results obtained from simulation runs that altered each of the three parameters in turn.

Our results show that the botmaster can balance the botnet latency and stealthiness by changing the parameters, called the *control variables*, under his control. He can achieve a high GoS even for a low desired response time. For example, if the bots receive messages at a rate given by a Poisson distribution with a mean at 500 messages per day, the GoS is above 90% for desired response time of 15 minutes (Figure 6a). For a specific desired time, the GoS decreases as the number of messages received per day increases. If bots are able to communicate their receipt rate M back to the botmaster, then the botmaster can adjust other parameters to maintain a desired GoS.

For example, the botmaster can control each bot’s CPU utilization. A high utilization would diminish stealthiness as the bot may become visible to the user of the victim machine. Figure 6b shows that even as the botmaster improves stealthiness by reducing the CPU usage of the bot, most of the bots respond within the desired time. For CPU utilization as low as 12.5%, GoS is about 80% within 1 hour. This GoS is reached in 2.5 hours for 5% utilization. The botmaster can further control the processing time at the host by changing the control variables l_k and l_m to achieve a high GoS while keeping CPU utilization low.

The botmaster can also achieve a higher GoS by controlling the time Δ between consecutive email checks by the bot. Δ can even be modified to reflect the email behavior of human users. A higher Δ value typically provides better stealthiness as many users check email infrequently. For a reasonable Δ value of 2 hours and with each bot checking for new mail at a time uniformly distributed within the 2 hour window, the GoS reaches 47% in 60 minutes and 80% within 100 minutes (Figure 6c).

These simulation results reinforce the theoretical discussion of the previous section. The results show that by manipulating values of control variables, the botmaster can

often achieve a target GoS within the desired response time for their particular botnet functionality.

8. Detection Mechanisms

Although the primary focus of our paper is on the design of a stealthy and hard-to-disrupt botnet C&C channel, our goal is to analyze a possible botnet innovation so that we can proactively develop defenses. We consider five different defensive strategies; each of our strategies does have some type of shortcoming. We expect that other researchers could develop alternative useful techniques.

A straightforward detection mechanism could operate at the end host rather than at the servers of an email provider. Just as the bots themselves leverage the smaller volume of emails at an individual email account rather than at a mail server, the detector can leverage the small volume. This approach distributes the detection load to end hosts and hence will scale well to large numbers of email users and large volumes of emails. Provided that the end host detector processes an email message before any other program on the host has access to the message, the detector can prevent bot commands from reaching the bot program. As a downside, an email provider would be unable to shield its customers from the compute costs of command detection.

Email messages encrypted and signed by the sender could serve as a solution. The receiver could drop any message whose sender’s identity could not be verified or whose text decrypts to non-recognizable content. A difficulty arises in solution if messages from new, legitimate senders have a high probability of being rejected. The complexity of key distribution among communicating parties may also impede adoption of this technique.

A “personal firewall” could detect accesses to an email account that were made by an unauthorized program. Similarly, host-based tools [10] that correlate outbound network connections to user input can be used to detect autonomous reading of emails by bots. Applications such as email clients periodically make outbound network connections to download email without requiring any user input and need to be whitelisted. A bot could use social engineer-

ing, perhaps by making itself look like a new mail notifier, that would entice users to whitelist the communication.

Defenders could disrupt the bootstrapping email account before the botmaster retrieves all bot information. Given the short lifetime of the account, this requires a quick response that may not be possible due to international boundaries and the time needed to analyze a bot binary.

When bots use brute-force decryption, ethically ambiguous solutions are possible. An email provider could intentionally send spam to their clients. The provider would not need to analyze their own generated messages, as they would obviously contain no embedded C&C commands. However, a bot on a client computer would process an increased volume of email. The botmaster would need to weaken the encryption key length so that bot can keep up with the rate of incoming email without overconsuming the CPU. Although centralized decryption at the email provider becomes less expensive, we do not expect any legitimate email provider to adopt a solution of this nature.

9. Conclusions

As a way to anticipate future trends in botnet design, we developed a stealthy C&C channel that distributes commands from the botmaster to bots via encoded email messages. Our blended messages cannot be efficiently identified by email providers even if the provider has a complete analysis of a captured bot, and hence is robust against detection and disruption. This paper was a proof-of-concept: we successfully showed that both blended spam and blended non-spam mail carried hidden commands, and we reasoned that a botmaster can make detection prohibitively expensive.

Our email-based C&C channel raises questions about the current handling of spam and strategies used to detect botnets. We showed that spam can become more than just a nuisance by carrying dangerous, hidden content. Perhaps email providers should consider accesses to spam differently than accesses to other email messages. The server could require additional authorization or human confirmation before allowing the client to access and download spam messages. The solution may need to be more aggressive. Given that malicious users are already encoding secrets in spam [14], it may be time to start considering exotic solutions such as aggressive anti-spam policies that drop higher volumes of spam than providers presently drop.

Successful email botnets provide a sign that current botnet detection strategies focusing on network-level detection will not remain effective in the long term. When communication becomes stealthy and impractical to identify within the network, detection may be easier at the endpoints. Rather than searching network flows for botnet communications, host-based malware detectors could search program images and process behaviors for evidence of bot programs.

Acknowledgement

This material is based upon work supported in part by the NSF under grants CCR-0133629, CNS-0627477, and CNS-0716570, and by the U.S. Army Research Office under grant W911NF0610042. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF and the U.S. Army Research Office.

References

- [1] FreePOPs. <http://www.freepops.org/en/>.
- [2] Linux Mailing List. <http://www.lwn.net>.
- [3] Phatbot trojan analysis. <http://www.lurhq.com/phatbot.html>.
- [4] Sinit P2P trojan analysis. <http://www.lurhq.com/sinit.html>.
- [5] M. Arrington. Single Ajax interface for Yahoo mail & IM coming, Nov. 2006. <http://www.techcrunch.com/2006/11/09/single-ajax-interface-for-yahoo-mail-im-coming/>.
- [6] P. Barford and V. Yegneswaran. An inside look at botnets. *Advances in Information Security*, 2006.
- [7] J. Blosser and D. Josephsen. Scalable centralized bayesian spam mitigation with bogofilter. In *USENIX Conference on System Administration*, Atlanta, GA, Nov. 2004.
- [8] M. Chapman and G. I. Davida. Plausible deniability using automated linguistic steganography. In *Conference on Infrastructure Security*, London, UK, Oct. 2002.
- [9] E. Cooke, F. Jahanian, and D. Mcpherson. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Workshop on Steps to Reducing Unwanted Traffic on the Internet (SRUTI)*, Cambridge, MA, June 2005.
- [10] W. Cui, R. H. Katz, and W. Tan. BINDER: An extrusion-based break-in detector for personal computers. In *USENIX Annual Technical Conference*, Anaheim, CA, Apr. 2005.
- [11] D. Dagon, C. Zou, and W. Lee. Modeling botnet propagation using time zones. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2006.
- [12] B. Eckman. <http://lists.sans.org/pipermail/unisog/2006-April/026261.html>.
- [13] L. International. Key logger woes: How this type of spyware can hurt you, 2006. <http://whitepapers.zdnet.com/whitepaper.aspx?docid=178232>.
- [14] C. Kret. Nobody's anonymous: Spam tracking and covert channels. In *Black Hat Windows Security Briefings*, Las Vegas, NV, July 2004.
- [15] G. Mori and J. Malik. Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA. In *Conference on Computer Vision and Pattern Recognition*, Madison, WI, June 2003.
- [16] J. Nazario, J. Anderson, R. Walsh, and C. Connelly. The future of internet worms. Technical report, Crimelabs Research, July 2001.
- [17] M. A. Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Internet Measurement Conference*, Rio de Janeiro, Brazil, Oct. 2006.
- [18] E. S. Raymond. *The Cathedral and the Bazaar*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 1999.
- [19] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk e-mail. In *Workshop on Learning for Text Categorization*, Madison, WI, July 1998.
- [20] R. Vogt, J. Aycocock, and M. Jacobson. Army of botnets. In *Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, Feb. 2007.
- [21] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: Using hard AI problems for security. In *EUROCRYPT*, Warsaw, Poland, May 2003.
- [22] P. Wang, S. Sparks, and C. Zou. An advanced hybrid peer-to-peer botnet. In *HotBots'07*, Cambridge, MA, Apr. 2007.