A Distributed Content Independent Method for Spam Detection

Alex Brodsky
University of Winnipeg,
Winnipeg, MB, Canada, R3B 2E9,
`abrodsky@acs.uwinnipeg.ca`

Dmitry Brodsky
Microsoft Corporation,
Redmond, WA, USA, 98033,
`dbrodsky@microsoft.com`

Abstract

The amount of spam has skyrocketed in the recent past. Traditionally, spam was sent by single source mass mailers (spammers), making it relatively easy to screen out through the use of blacklists. Recently spammers started using botnets to send out the spam, rendering the blacklists ineffective. Although, content-based spam filters provide temporary relief, this is a never-ending cat-and-mouse game between spammers and filter developers.

We propose a distributed, content independent, spam classification system that is specifically aimed at botnet generated spam and can be used in combination with existing spam classifiers. Our proposed system uses source identification in combination with a peer-to-peer based distributed database to identify e-mails that are likely to have originated from botnets. The system is distributed in order to provide a robust defense against denial-of-service attacks from the very same botnets. Lastly, our system is specifically designed to be used within the existing e-mail infrastructure. It does not require special hardware, changes to the underlying protocols, or changes to the mail transfer agents.

1 Introduction

The amount of junk e-mail, commonly called spam, has skyrocketed in the recent past. Traditionally, spam was sent by single source mass mailers (spammers). This spam is relatively easy to screen by using lists of known spammers [5, 10, 15, 28, 29], and dropping all e-mail that originates at a listed address. Recently, the spammers started using distributed networks of hijacked PCs, called botnets [6, 8], to send spam, rendering the list system ineffective.

Although content-based filters provide some temporary relief from spam, they represent a cat-and-mouse game between the spammers and the spam filter developers—every time the developers improve their filters, the spammers create new ways of fooling them. Consequently, a content-independent method is needed.

In the past, spammers used nondistributed systems that in turn were susceptible to nondistributed countermeasures. The advent of botnets has given the spammers a distinct advantage, rendering many of the existing (nondistributed) countermeasures ineffective. To rectify this asymmetry, distributed countermeasures are needed.

We propose a distributed, content independent, spam classification system, called *Trinity*, that is specifically aimed at botnet generated spam and can be used in combination with existing spam classifiers. In order to be viable, Trinity must be easy to integrate within the existing e-mail infrastructure. Specifically, it must not require changes to existing protocols or mail transfer agent (MTA), it can easily be hosted on the same host as the MTA, and it must work in combination with existing spam filtering mechanisms. Secondly, the system should provide the recipients with the option to bypass Trinity. Third, the system must be resilient to denial-of-service attacks, and must not have single point of failure modes, such as centralized servers.

Trinity is based on the following observation: in order to be effective, bots must send a relatively large number of e-mails in a short amount of time. This stems from the fact that most computers in a botnet are only up for a short period each day; in the evenings when the average user comes home and turns on their machine. Consequently, if an e-mail is received from an "unknown" source that has sent many e-mails in a short period of time, then the likelihood of this being spam is high.

Trinity receives each e-mail from the MTA via an existing framework such as SpamAssassin [27]. It first determines the source of the e-mail (IP address) using the e-mail's envelope—a challenging task in itself [11]. Using this IP address, Trinity updates and queries the distributed database, which is updated by all MTAs that use Trinity. The database tracks e-mail sources and the number of e-mails that the source recently sent within a fixed period—on the order of an hour. This score is added to the e-mail's envelope, in the form of an `X-` line, and may also be returned to the general spam classification framework. The score is then used by the mail user agent (MUA), typically called a mail client, to classify the e-mail. Namely, if the score is high, and the sender is not in the recipient's address book, or the sender/recipient[1] lists, then the e-mail must be spam. This late-stage classification, i.e., at the mail client, prevents incorrect classification of e-mails originating from mailing lists.

The remainder of this paper is divided as follows. The next section describes the problem and challenges that Trinity addresses. Section 3 provides an overview of Trinity's design assumptions, its architecture, and the technical challenges involved. Sections 4, 5, and 6 describe the design of Trinity's components, and how they address some of the technical challenges. Section 7 compares Trinity to existing e-mail classification approaches and Section 8 summarizes and discusses future work.

2 Botnet Generated Spam

In the past spam came from static sources such as marketing companies, various e-commerce firms, and third-party mailers. In these cases, the spam was sent from a single source with a fixed IP address. E-mail originating at these sources could easily be classified by checking its source against a blacklist of known spammers [5, 10, 15, 28, 29]. Provided that the blacklists are correct and up-to-date, classifying spam that originates at a single source is straightforward.

Recently, to circumvent blacklists, spammers began using botnets. Botnets, are networks of bots, which are computers that are hijacked via a virus, worm, Trojan, or some other malware infection. Typically, the bot is connected to the Internet via a DSL or cable subscription and is assigned an IP dynamically, at boot time. Since most users only turn on their machines when they need to use them, the bot's IP address can change on a daily or even hourly basis due to DHCP churn [22]. Once, a bot is activated it can be remotely instructed, to infect other machines, participate in denial-of-service attacks, or send spam.

In the latter case, the bot can use the ISP's (Internet Service Provider) own SMTP server to relay the spam. Since the server cannot differentiate between e-mail sent by the actual user from that sent by the bot, it becomes an unwilling accomplice in the distribution of spam. Furthermore, since the IP address of the bot can change from day to day, blacklists are rendered ineffective. Adding the relay to the blacklist cannot be done since it would also block all legitimate e-mails from the entire ISP.

In response to these tactics, commercial mail systems such as IronPort [14] and CipherTrust [4] have started quarantining or aging received e-mail to allow the blacklists time to catch up. In particular, this is done when the number of e-mails from the same source spikes. However, this is problematic for a couple reasons. First, e-mail is delayed for long periods of time, eliciting complaints from users. Second, as was noted in [22, 17], the average bot sends on average 10 spams per day to a specific domain. Thus, the chances of the mail transfer agent receiving many e-mails from a single host is relatively small.

A different approach to identifying spam is to classify e-mail based on its content i.e., the body of the e-mail. This approach does not rely on the source of the e-mail being static and does not suffer from the lag between when a new spammer commences operations, and when it is added to a blacklist. Unfortunately, the result is a cat-and-mouse game between the spammers and the filter developers. Every time new filters are developed, spammers develop new countermeasures, forcing the filter developers to create new filters. This not only uses up significant resources, but also increases the potential for false-positives; given the importance that e-mail has acquired in our daily lives, this is becoming less and less tolerable. Furthermore, it takes much more time to develop or tune a filter than to add a host to a blacklist.

Several collaborative and distributed e-mail classification approaches have been tried [3, 9, 13, 19, 21, 31, 32]. These approaches keep a distributed database of known spam signatures, which are matched against incoming e-mail. However, these approaches depend on the same spam message being sent to many recipients. The Achilles heel of these approaches is that an e-mail messages can easily be tailored to each recipient. In fact, any content based approach—relying on the contents of the e-mail—is due to the same fate, an endless cat-and-mouse game.

Any viable approach to this problem must surmount several challenges. First, it must be easily installable and pluggable within the existing infrastructure. Namely, no existing protocols or software need be modified, it should use existing frameworks and interfaces, without modifications to the mail transfer agents or mail clients.

Second, it must not rely on the contents of the e-mail, just the envelope. Third, the system must scale, be secure, and have no central point of failure. Particularly, it must not be susceptible to malicious hosts that masquerade as valid MTAs and must be resistant to distributed denial-of-service attacks that can be launched by botnets. Fourth, it must be user-controllable, i.e., it can be disabled by the recipient. Lastly, the system has to respond rapidly to spam. Since the IP address linking a bot to the e-mail it sends is temporal, the system must identify the source of a spam session quickly and ensure that stale information is not retained. Our system, Trinity, meets these criteria.

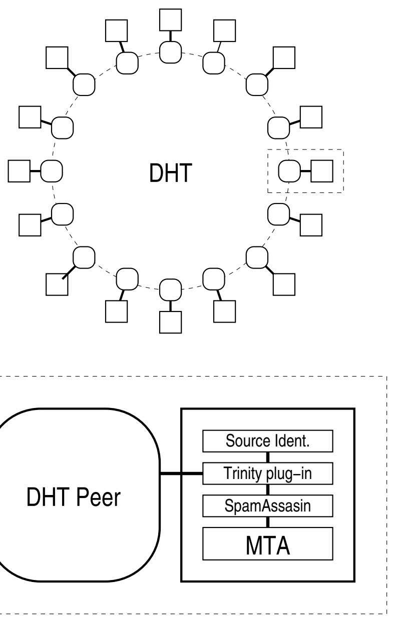
3 Overview of Trinity

Trinity is based on the assumption that there is a large class of spam-bots that send large amounts of e-mail over short periods of time. For example, approximately 70 billion e-mails are sent each day, 70%[2] of which are spam [28]. Even if 20% (10 billion) of the spam is sent by bots, and there are (conservatively) 150 million bots, then on average each bot must send at least 67 spams a day. Since it is believed that a large fraction of bots send very few spams each day [22], there must exist a significant fraction of bots that send hundreds of spams per day. Given that the average user only keeps their computer on for a few hours each day, such a bot must therefore send 50 or more spams per hour to meet its quota.

A host that has recently sent large amounts of e-mails may be a spam-bot. Consequently, any e-mail coming from such hosts is potentially spam, and if the source has a dynamically allocated IP address[3] (or simply a *dynamic IP address*)[4] and the sender is not in the recipient's address book or list of past recipients or senders, then it is almost certain that the e-mail is spam.

Since a spam-bot does not necessarily target multiple recipients within a single domain, determining whether a bot is sending spam is inherently a collaborative effort. Trinity, is a distributed spam detection system that identifies the source of each e-mail and then stores this information in a distributed database that is used and updated by all peers. Trinity depends on the collaboration of peers: as the number of peers increases, spam-bot identification improves because the sample size increases. In essence, an effective countermeasure to distributed spamming requires a distributed effort.

Classification consists of several distinct parts: identifying the source of e-mails, keeping track of how many e-mails were recently sent by a source, and disseminating this information for the purposes of classifying fu-



ture e-mails. Additionally, these tasks must be coordinated for each e-mail as it is received by an MTA. Thus, Trinity comprises four parts that perform the respective functions.

When an e-mail arrives at an MTA, the message is passed to a general spam classification framework such as SpamAssassin [27]. The message is passed to a variety of plug-ins (within SpamAssassin), including a Trinity plug-in, which is responsible for coordinating the source identification and source tracking tasks, and embedding the resulting classification in the e-mail.

The plug-in first passes the message envelope to the source identification server, a locally running process, that determines the source of the e-mail—a challenging task—and whether the source of the e-mail is a static or dynamic IP address.[5] If the source has a static IP address, we can use the existing blacklist mechanisms, i.e., if its source is listed in one of the blacklists, it can immediately be classified as spam. In this case, no further processing is necessary. Otherwise, if the source has a dynamic IP address, the address is returned to the plug-in for the next stage.[6]

Second, the plug-in updates the distributed database used to track e-mail sources. A local server, which is part of the distributed database, is assumed to run on the same host as the MTA or on a nearby host. The server propagates the update to the distributed database, stores chunks of the distributed database, and caches updates and queries for the local MTA.

Third, the plug-in queries the distributed database, via the local server, about the number of e-mails that were recently sent, typically within an hour, from the same source as the current e-mail—this is the *sender score*. For efficiency reasons, this query may occur as part of the database update. If the sender score is high, i.e., many e-mails were sent, then the score is appended to the e-mail's envelope and returned to the spam framework itself. If the score is low, this indicates that either the source has not sent many e-mails recently, or that the e-mail may be one of the first of many e-mails that were sent. To distinguish these two cases, the plug-in can quarantine the message for a short period of time, and then perform the query again. The score from the second query is then appended to the e-mail's envelope and returned to the framework. In our system the quarantine has minimal impact because it is only used for e-mails arriving from senders with dynamic IP addresses, and the length of the quarantine is on the order of minutes, not hours, as in other systems [4, 14].

At this point the back-end processing by Trinity is complete. The e-mail, with its appended envelope, is stored to be consumed by a mail client. Modern mail clients have a highly configurable rule-based filter that can be used to complete the spam classification. Specifically, a simple rule can be used to determine if an e-mail with a high score is from an unknown sender—a sender is unknown if it is neither in the address book, or in the sender or recipient lists. If so, the e-mail can be junked.

The key technical challenges of Trinity is to correctly identify the source of an e-mail, to quickly and efficiently update the distributed database, to ensure that the database is not susceptible to poisoning from malevolent peers or to denial-of-service attacks, and to ensure that the system scales well, is easy to install and maintain, and does not require excessive resources.

As described in [11] determining the true source of an e-mail is difficult because the sender or any malicious mail relay can add false *Received* lines to the e-mail's envelope—each relay through which an e-mail travels must prepend a *Received* line to the e-mail's envelope, indicating the host from which the relay received the e-mail. The *Received* line prepended by the first trusted relay that received the e-mail contains the IP address of the host from which it received the e-mail. This host is considered to be the source of the e-mail. However determining the first trusted relay is challenging.

Although each Trinity installation could maintain its own complete database, this is not feasible—even though the database would only be a couple gigabytes in size. A database is useless unless it is promptly updated by other Trinity installations whenever they receive new e-mails. Since the number of e-mails sent per day is on the order of 70 billion [28] and a significant fraction of these e-mails would generate updates, a server that hosts the entire database, and the connection to it, would be swamped.

Trinity uses a distributed hash table (DHT), such as Chord [30], as its database. This is ideal, since the updates are commutative, may occasionally be lost, and the update latency is on the order of seconds. To minimize the amount of traffic generated by updates or queries Trinity uses UDP for both, especially since both the update and the query contain very little data.

Lastly, it is likely, that spammers would try to develop countermeasures against Trinity. The three main approaches would be to fool the source identifier, to poison the distributed database via false updates and malevolent peers in the distributed database, and to launch denial-of-service attacks against the distributed database with the very same botnets. The first countermeasure must be dealt with by the source identifier. The latter two countermeasures must be considered when designing the distributed hash table. These issues must be addressed to ensure that Trinity remains effective in today's Internet environment. In the next sections we discuss how some of these issues can be addressed.

4 E-mail Source Identification

In a perfect world, the source of an e-mail can easily be identified from the *Received* lines as described in RFC 2821 [18]; these lines are found in the envelope of the e-mail. Every time the e-mail passes through a SMTP relay, the relay must prepend an additional *Received* line to the list of *Received* lines and must not modify any of the other *Received* lines [18]. Each *Received* line must identify *from* which host the e-mail was received and *by* which host the e-mail was received. Thus, the *Received* lines form a kind of a linked list that links the current recipient of the e-mail to the sending host—in theory.

Unfortunately, there is nothing to prevent a malicious sender from adding one or more *Received* lines to the envelope. Furthermore, if an e-mail passes through a malicious relay, then all *Received* lines may be modified or deleted. Fortunately, all is not lost.

We must assume that once the e-mail is received by a trusted relay, say that of an ISP, that all relays through which the e-mail passes post-hence are also trusted. Thus, the *Received* lines added by these relays are correct. That is, the *Received* line added by the first trusted relay, identifies the untrusted host that injected the e-mail. The host may or may not be malicious, but for our purposes, the host is considered the source of the e-mail.

The key challenge then is to identify this first trusted relay, and its corresponding *Received* line, which contains the de facto IP address of the sender. As was pointed out in [11] this is not easy and in fact does not have a deterministic solution. Fortunately, the problem domain provides a reasonable heuristic to this problem. Recall, that the reason botnets are a problem is because the bots typically do not have static IP addresses. Thus, we consider a relay trusted if it has a static IP address.

Typically, most relays are configured to receive e-mail only from hosts that have a valid domain name and reverse DNS lookup entries. The main exception are relays that receive e-mail from the ISP's clients and relays it to its destination. In this case, the relays only accept e-mail from the ISP's subnet and in many cases require client authentication. These relays are the first trusted relays in the delivery chain.

Trinity's source identification process comprises two phases: a preparatory phase and a traversal phase. The first phase identifies the point from which the traversal of *Received* lines is to begin. In many cases, once an ISP's ingress relay—the one listed in the ISP's MX DNS entry—receives an e-mail, it passes the e-mail through zero or more additional relays before storing and processing it. Since there is no need to check those relays, the natural starting point is the ISP's ingress relay—all the relays prior to the ingress relay need to be checked. Consequently, when a Trinity peer starts up, it must determine the IP address of the ingress relay.

Unfortunately, this information cannot always be determined from the MX entry [11] because the entry may refer to a load-balancing router that distributes the SMTP connections between several relays. There are several approaches: First, if there is only one ingress relay, then the MX entry is sufficient. The second option is to provide an explicit list of ingress relays. The third option is to perform a kind of tomogram of the ingress relays by asking a known (source) relay to probe the ISP's ingress relays. The source relay sends e-mails with special *X-* lines to the mail exchanger listed in the ISP's MX record. These *X-* lines allow Trinity to separate these probes from regular e-mails and extract the *Received* line that identifies an ingress relay. Since the source relay is known, the *Received* line in the probes' envelope that identifies the source relay also identifies one of the ISP's ingress relays. A sufficient number of probes will identify, with high probability, all the ingress relays. In all likelihood though, since the ISP's administrators are responsible for configuring both the ingress relays and the source identifier, the former two approaches should prove sufficient. Potentially, SPF [34] or DKIM [1] may also be used to identify the relay that contacts the ingress relay, making identification of the ingress relay unnecessary.

Assuming that the IP addresses of the ingress relays are known, the preparatory phase simply scans the *Received* lines from top to bottom, until a line that lists an ingress relay is found. This is sufficient to complete the preparatory phase.

The traversal phase begins at the *Received* line corresponding to the ingress relay, and checks if the relay from which the email was received has a static or dynamic IP address. To do this several heuristics are used and we are investigating others as well. These heuristics include looking up the IP address in Policy Block Lists (PBLs), such as the one maintained by Spamhaus [29], performing a reverse host lookup on the IP address, checking whether the listed name, matches the one in the *Received* line, and analyzing the name itself. For example, in many cases, names that are permanently associated with dynamic IP addresses contain all or part of the IP address in the name, as well as indicators, e.g., the word "dynamic". Note, there is currently no fool-proof way to distinguish between dynamically and statically allocated IP addresses. To reduce DNS queries previously identified IP addresses are cached.

If the host listed in the current *Received* line is determined to have a dynamic IP address, then the traversal phase completes and the resulting IP address is deemed to be the source of the e-mail. Otherwise, the traversal phase continues to the next *Received* line.[7] The traversal phase terminates if it runs out of *Received* lines—in which case the source has a static IP address—or a host with a dynamic IP address is found, i.e., an untrusted relay, which is then assumed to be the source of the e-mail. At this point, the source identification is complete and the database is updated. Note: We are currently in the process of designing more heuristics, and intend to implement and test their efficacy.

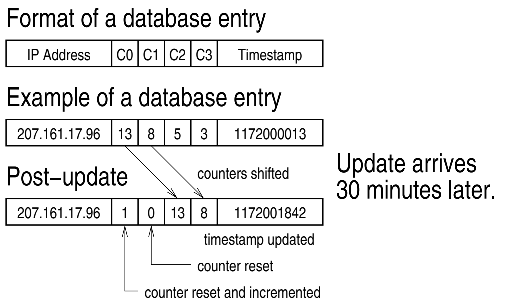
As frameworks such as the Sender Policy Framework (SPF) [34], or the DomainKeys Identified Mail (DKIM) [1] are adopted, it may become easier to identify the source of the an e-mail, thus obviating the need for ad-hoc source identification.

5 The Distributed Database

The distributed database is implemented using a distributed hash table, such as Chord [30]. First, the database must handle billions of updates per day: even though the updates are small, on the order of five bytes, any centralized solution would be overwhelmed by the sheer numbers. Second, hosts receiving the e-mails must quickly and efficiently query and update the database. Delays in receiving or processing the updates would render the system ineffective because peers could not effectively gauge the number of e-mails that a host has recently sent.

The database itself is relatively small, consisting of entries that comprise about 12 bytes: the IP address, four 8-bit counters, and a local timestamp that indicates the last time that the counters were updated. The four counters represent the number of e-mails sent by the host with the corresponding IP address. Each counter denotes a quarter of an interval, say an hour. That is, counter 0 stores the number of sent e-mails in the last 15 minutes, counter 1 stores the number of sent e-mails in the preceding 15 minutes, and so on.

Consequently, it is trivial to update the counters. First, compute the number of interval quarters that have passed since the last update by using the current time and the associated timestamp—say s quarters have passed. Shift the counters appropriately, $C_i = C_{i-s}$, $i \geq s$, where C_0, C_1, C_2, C_3, are the four counters. Reset the remaining counters, $C_i = 0$, $i < s$. Increment the counter C_0,[8] and update the timestamp. The database can even be pruned by a low priority process that removes entries with expired timestamps.



This database is distributed among the peers that participate in the DHT, where each peer is responsible for entries containing IP addresses that map to it. Furthermore, these entries are forwarded to k of the peer's neighbours, a simple form of replication. Since the full database would be at most a few gigabytes in size, each peer would store on the order of 10 to 50 megabytes of data (depending on number of participating peers), and could keep a working copy in memory at all times.

An update to the database originates from the Trinity plug-in at an MTA. The update consists of the source IP address and a counter indicating how many e-mails were received from the host.[9] The update is sent to the local peer that determines which peer is responsible for storing the entry, based on the IP address, and forwards the entry. Each peer caches other peers' addresses for efficient operation. All updates and queries are sent via UDP, since a few lost updates does not adversely affect the system. Once the update arrives at the destination peer, it is applied to the entry, and forwarded to the neighbour peers via permanent TCP connections. Compared to the cost of setting up and tearing down a TCP connection to send an e-mail, the cost of an update is small. Since an update simply increments a counter, updates are commutative, meaning that the order in which they are applied doesn't matter.

The key challenge is to ensure that the update is delivered and applied within a short period of time. Since each peer can keep the entire database in memory, and the cost of applying an update is negligible, the primary latency is delivery of the update. Since this takes $O(\log N)$ sends, where N is the number of peers, it is difficult to imagine a situation where the update is not received in under a minute. Furthermore, since updates are commutative, the order in which they arrive does not matter. Since consistency is not required, the cost of replication, i.e., forwarding the update to the neighbour, is also small.

To reduce the number of updates several heuristics can be used. The simplest heuristic is to cease updates for spam-bots that have sent beyond a given threshold. For example, if spam-bot has sent 100 e-mails in the last 10 minutes, there is little point in sending an update about the 101st e-mail, particularly if any host that sends more than 30 e-mails per hour is likely to be a spam-bot. Another heuristic is to delay updates for a short period of time to see if the same host sends to the same MTA. In this case the counter in the update is incremented. However, to ensure that the database is promptly updated, the update should be sent within a short period of time.

Although it may seem like a good idea to map all IP addresses from the same network to the same set of peers, this creates a denial-of-service target, since the spammers can then selectively attack parts of the DHT to mask botnets in the corresponding network. This leads us to the next challenge, how to deal with malevolent peers and other countermeasures against Trinity.

6 Securing Trinity

To be effective Trinity must deal with two types of threats. Database poisoning by malevolent peers and distributed denial-of-service attacks (DDOS). We consider the latter threat first. To mitigate the usefulness of a

DDOS attack the entries are mapped to peers in such a way such that no peer stores a significantly large fraction of IP addresses from a particular network or region. Consequently, a DDOS attack would have to attack a significant fraction of the DHT in order to affect its functionality. The decentralized nature of the DHT is its best defense against DDOS attacks.

If a peer is attacked, its neighbours take over for it, since each of them has a replica of the database fragment for which the peer was responsible. Furthermore, since Trinity only relies on recent data, neighbouring peers that have not replicated the fragment can completely rebuild the database within an hour, one interval, because all data in the afore mentioned fragment becomes stale. The short life span of the data obviates any need for complicated and expensive replication procedures.

In some cases, for particularly poorly or improperly configured networks, it is possible for fake updates or queries to be sent with fake source addresses. To mitigate this problem, all queries and updates, sent from peer to peer, simply include a small pairwise shared secret, that is changed at regular intervals. Thus, false queries and updates can be discarded. If the malevolent sender uses its true IP address, the IP can be added to the peer's firewall's blacklist, dropping all packets from the sender. Queries and updates to the local peer from the Trinity plug-in are done over a secure network segment, a permanent and secure TCP connection over a local segment, or over a local connection if the peer and the MTA are running on the same host.

Even if a malevolent update is successfully delivered, all it does is increase the count of e-mails that some host has sent. That is, it may increase the chance of a false-positive, but cannot increase the chance of a false-negative, which would be the goal of a spammer.

Unfortunately, there is little to prevent a spammer from setting up their own peer and joining the DHT. Basic checks, such as ensuring that the peer is not in any of the spam blacklists and that the peer has a static IP address, eliminate the less determined spammers. However, no automated system can fully separate friend from foe. Consequently, it is necessary to assume that malevolent peers will participate in Trinity and will be part of the DHT.

This needs to be addressed at two levels. First, how to protect the DHT from malevolent peers. Second, assuming the DHT is protected from malevolent peers, how to protect the Trinity database and its functions from being affected by malevolent peers. There has been a significant amount of research into security issues of peer-to-peer systems and the corresponding solutions [2, 7, 20, 26, 33]. Roughly speaking, the issues can be categorized either as as routing issues, where malevolent peers interfere with the delivery of an update or a query; storage and retrieval issues, where malevolent peers do not store, deny that they are storing, or falsify data that they are responsible for; and denial-of-service attacks that divert system resources [26]. Since these issues are addressed in [2, 7, 20, 26, 33], we do not discuss them here due to space considerations. However, we note that one distinguishing feature of Trinity's use of DHTs is that the peers are expected to be up for extended periods of time, that is, the number of joins and leaves is expected to be small. Consequently, Trinity can take extra time to validate new peers before allowing them to participate in the DHT. Furthermore, since virtual hosts are not useful for this application, they are disallowed. These restrictions mitigate many of the security concerns.

Assuming that the DHT is functioning, the next challenge is to ensure that malevolent peers do not have a significant effect, i.e., that a malevolent peer cannot withhold or falsify data for extended periods of operation. This can be accomplished using two mechanisms: replication and reputation. First, we note that the updates are replicated across a group of peers. Thus, instead of sending the update or query to just a single peer, the query or update can be sent to several peers, randomly chosen from the group. The results of the queries can be compared, against each other and against previous queries to ensure that the responses are consistent. At this point the reputation mechanism is used. The DHT can also be used to store a reputation counter for each peer. If a peer returns inconsistent responses, then the peer's reputation is decreased. The reputation is restored using the same multicounter decay mechanism as described in the previous section. If the reputation of a host drops below a threshold, it is excommunicated from the DHT. The key point is that no single peer is trusted with a fragment of the database. Instead, a group of peers is trusted with fragment. The groups comprise a small number of peers whose addresses are adjacent in the DHT, meaning that a peer belongs to several groups. Since the chance of two or more malevolent peers being in the same group is negligible (for DHTs with sufficiently many peers), this effectively ensures that malevolent peers cannot significantly diminish the function of the distributed database.

The only other attack that malevolent peers could do is send false updates, increasing the counts of various hosts. However, since this can only increase the estimate of how many recent e-mails a host has sent, it does not diminish the ability of Trinity to identify potential spambots. In theory, increasing the estimate of how many e-mails a host sends can increase the false-positive rate. However, to be noticeable, the malevolent peer or peers must increase the estimates of many hosts. This would require the malevolent peers to send updates at a noticeably faster rate. Furthermore, since these estimates decay rapidly, the malevolent peers would have to continuously send false updates. By tracking, as part of their reputations, the rate at which peers send updates, malevolent peers can be identified and excommunicated.

7 Related Work

Traditional spam classifiers fall into one of two categories: list-based classification, such as the blacklists published by Spamhaus [29], and content-based classifiers that analyze the content of the e-mail to classify it—commonly used methods include Bayesian filters [12, 23] and pattern matching [25]. The blacklists [5, 15, 10, 28, 29] are either static lists that are periodically updated by downloading new ones, or are stored in remote databases, which are themselves updated on a regular basis. A common technique, DNS Blacklists (DNSBL) [15], uses the DNS system to store the blacklists and serve the queries. Typically a mix of blacklists is used [5, 16, 24], depending on the desired level of aggressiveness and tolerance for false-positives. The effectiveness of blacklists has been investigated in [17, 22]. While blacklists are effective against hosts with static IP addresses, their response time is much too slow to counter botnets.

Content-based classification analyzes the contents and envelope of an e-mail using Bayesian networks [12, 23] or pattern matching [25]. Such methods work well against known spam. I.e., spam that has been seen in the past and contains known strings or patterns, e.g., advertising the impotence drug of the month. Unfortunately, the majority of e-mail clients now render Hypertext Markup Language (HTML) based e-mails, allowing spammers many opportunities to fool the filters. Some examples, include using embedded pictures, clever use of different font sizes, and foreground and background colours. Content-based filters require never-ending tuning and adjustment in order to keep up with the spammers' latest tricks.

One approach to improve the responsiveness of the content-based filters is to use a distributed database of known spams [3, 9, 13, 19, 21, 31, 32]. When a peer in these systems receives an e-mail that it (or a user) classifies as spam, the distributed database is updated with a signature of the e-mail's contents. When an MTA receives a e-mail it computes the e-mail's signature and queries the database. If the signature is in the database, the e-mail is classified as spam. The problem with these approaches is similar to that of stand-alone content-based classification. Namely, that spam-bots can be programmed to sufficiently alter the content of each e-mail such that each e-mail yields a unique signature, even though they look the same.

Another approach is to adjust the e-mail system by using Sender Policy Framework [34] (SPF), which is an extension to SMTP [18]. This scheme uses the DNS system to specify which hosts may relay e-mail from a given domain[10]. However, this system is ineffective against botnets, since they typically use the domain's own relay which cannot distinguish between a host sending spam and one sending a real e-mail. Another approach is DomainKeys Identified Mail (DKIM) [1], which associates a "responsible identity" with each e-mail. Allowing the receiver to confirm the sender and origin of the email. Unfortunately, this system does not prevent the bot from using the identities stored on the hijacked computer and sending email through the domain's relays. It does however, make it easier to identify the source of the email. Adoption, as in many other cases, may prove to be the biggest hurdle for DKIM. Other approaches include forcing ISPs to police their users, requiring SMTPS authentication, and rate-limiting the number of e-mails that a host can send. Unfortunately, these approaches require the ISPs to incur additional cost and compliance is optional, meaning that many ISPs will not do this to avoid creating additional inconveniences or hassles for their customers.

8 Discussion and Future Work

In this paper we described Trinity, a system that is designed to identify spam originating from botnets. Trinity is designed to be used in concert with existing spam classifiers and complements, rather than replaces, the use of blacklists. The system is content-independent and does not require additional investment in new infrastructure or technology. Furthermore, Trinity is resilient to denial-of-service attacks.

8.1 Deployment

Initially, the number of Trinity peers will invariably be small. Since Trinity is a distributed system, this may affect Trinity's functionality with respect to load distribution, security, and coverage. First, the load on Trinity, the rate of updates, is generated by the peers. Hence, the load is proportional to the number of peers and therefore scales with the Trinity's size.

Second, Trinity uses groups of peers to monitor each other to ensure security. Initially, when there are few hosts, it may not be possible to form sufficiently large groups. However, at this stage, it is still possible to manually verify, admit, and monitor potential peers. Once the system has a sufficient number of peers, to form sufficiently large groups, the system can begin to automatically admit and monitor its peers. Furthermore, during Trinity's deployment it is less likely to present a target since its coverage is limited by its size.

Third, Trinity relies on the collaboration of MTAs to accurately gauge the number of e-mails coming from a particular source, the primary challenge is to achieve the necessary critical mass of peers—a standard problem with any new technology. To facilitate this critical mass, the Trinity plug-in can be used in "light mode". In this mode the plug-in collects the *Received* lines from incoming envelopes, sanitizes them, and forwards the lines, via UDP, to a remote peer that performs source identification and updates the database. Such an approach would increase Trinity's coverage at a faster rate than its adoption. The plug-in, running in "light mode", can be included as part of a typical SpamAssassin [27] deployment.[11]

8.2 Future Work

We are presently in the process of implementing Trinity. Several issues still require investigation, such as the heuristics for performing source identification and the right trade-off between the security and weight of the protocols. One important question is how many peers must Trinity have before it is effective.

Acknowledgments

This work was supported by the National Science and Engineering Research Council of Canada (NSERC) Discovery research grant. We thank the referees and our shepherd for all their helpful comments and suggestions.

References

[1] ALLMAN, E., CALLAS, J., DELANY, M., LIBBEY, M., FENTON, J., AND THOMAS, M. Domainkeys identified mail (dkim) signatures. http://www.ietf.org/internet-drafts/draft-ietf-dkim-base-10.txt, 2007.

[2] CASTRO, M., DRUSCHEL, P., GANESH, A., ROWSTRON, A., AND WALLACH, D. Secure routing for structured peer-to-peer overlay networks. In *Proc. of the 5th ACM Symposium on Operating System Design and Implementation* (2002).

[3] CHUNG, A., TARASHANSKY, I., VAJAPEYAM, M., AND WAGNER, R. Spamstrangler: A chord-based distributed spam detection tool. Tech. Rep. http://pdos.csail.mit.edu/6.824-2002/projects/spamstrangler.ps, Massachusetts Institute of Technology, 2001.

[4] Ciphertrust. http://www.securecomputing.com/, 2007.

[5] Composite blocking list. http://cbl.abuseat.org/, 2007.

[6] COOKE, E., JAHANIAN, F., AND MCPHERSON, D. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Usenix Workshop on Steps to Reducing Unwanted Traffic on the Internet* (2005).

[7] DABEK, F., BRUNSKILL, E., KAASHOEK, M. F., KARGER, D., MORRIS, R., STOICA, I., AND BALAKRISHNAN, H. Building peer-to-peer systems with Chord, a distributed lookup service. In *Eighth IEEE Workshop on Hot Topics in Operating Systems* (2001).

[8] DAGON, D., ZOU, C., AND LEE, W. Modeling botnet propagation using time zones. In *Proc. of the 13th Annual Network and Distributed System Security Symposium* (2006).

[9] DAMIANI, E., DE VIMERCATI, S. D. C., AND SAMARATI, P. P2P-based collaborative spam detection and filtering. In *Proc. of 4th IEEE Conference on P2P* (2004).

[10] Distributed sender blocking list. http://dsbl.org, 2007.

[11] GOODMAN, J. IP addresses in email clients. In *First Conference on Email and Anti-Spam* (2004).

[12] GRAHAM, P. A plan for spam. In *Reprinted in Paul Graham, Hackers and Painters, Big Ideas from the Computer Age, O'Really, 2004* (2002).

[13] GRAY, A., AND HAAHR, M. Personalised, collaborative spam filtering. In *Fourth Conference on Email and Anti-Spam* (2007).

[14] Ironport. http://www.ironport.com, 2007.

[15] IVERSON, A. Dnsbl resource. http://www.dnsbl.com/, 2007.

[16] IVERSON, A. Dnsbl statistics. http://stats.dnsbl.com/, 2007.

[17] JUNG, J., AND SIT, E. An empirical study of spam traffic and the use of DNS black lists. In *Proc. of the 4th ACM SIGCOMM Conference on Internet Measurement* (2004).

[18] KLENSIN, J. RFC 2821 – Simple Mail Transfer Protocol. http://tools.ietf.org/html/rfc2821, 2001.

[19] KONG, J., BOYKINY, P., REZAEI, B., SARSHAR, N., AND ROYCHOWDHURY, V. Scalable and reliable collaborative spam filters: Harnessing the global social email networks. In *3rd Annual Workshop on the Weblogging Ecosystem: Aggregation, Analysis and Dynamics* (2006).

[20] KUNZMANN, G., AND BINZENHOEFER, A. Autonomically improving the security and robustness of structured P2P overlays. In *International Conference on Systems and Networks Communications* (2006).

[21] PRAKASH, V. Vipul's Razor. http://razor.sourceforge.net/, 2007.

[22] RAMACHANDRAN, A., DAGON, D., AND FEAMSTER, N. Can DNS-based blacklists keep up with bots? In *Third Conference on Email and Anti-Spam* (2006).

[23] SAHAMI, M., DUMAIS, S., HECKERMAN, D., AND HORVITZ, E. A bayesian approach to filtering junk e-mail. In *AAAI-98 Workshop on Learning for Text Categorization* (1998).

[24] SenderBase. http://www.senderbase.org, 2007.

[25] SHOWALTER, T. RFC 3028 – Sieve: A Mail Filtering Language. http://tools.ietf.org/html/rfc3028, 2001.

[26] SIT, E., AND MORRIS, R. Security considerations for peer-to-peer distributed hash tables. In *International Workshop on Peer-to-Peer Systems* (2002), vol. 2429 of *Lecture notes in computer science*.

[27] SpamAssassin. http://spamassassin.apache.org/, 2007.

[28] SpamCop. http://www.spamcop.net/, 2007.

[29] Spamhaus. http://www.spamhaus.org/, 2007.

[30] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. of the ACM SIGCOMM 2001 Conference* (2001).

[31] TOBIN, F. Pyzor. http://pyzor.sourceforge.net/, 2007.

[32] VIXIE, P., AND RHYOLITE LLC. Distributed Checksum Clearinghouse. http://www.rhyolite.com/anti-spam/dcc/, 2007.

[33] WALLACH, D. A survey of peer-to-peer security issues. In *ISSS* (2002), M. Okada, B. Pierce, A. Scedrov, H. Tokuda, and A. Yonezawa, Eds., vol. 2609 of *Lecture Notes in Computer Science*.

[34] WONG, M., AND SCHLITT, W. RFC 4408 – Sender Policy Framework (SPF) for Authorizing Use of Domains in E-Mail, Version 1. http://tools.ietf.org/html/rfc4408, 2006.

Notes

[1] Modern mail clients track email addresses to which the recipient has sent email and from which the recipient has received valid email.

[2] Some estimates are much higher.

[3] Spamhaus [29] keeps a Policy Block List (PBL) of IP addresses that are known to be dynamically allocated.

[4] A host with a statically allocated IP address is said to have a *static IP address*.

[5] We do not foresee any difficulties with private or internal (back-net) IP addresses. These addresses are typically used for home or small office networks. In this case, the source is considered to be the entire private network whose single public IP address is assigned by its ISP.

[6] Keeping track of the number of emails sent by a source with a static address makes no sense, since hosts with static addresses, unlike those with dynamic addresses, are likely to be multiuser systems that send many e-mails in a short period of time.

[7] Note: we assume that e-mail from sources with static IP addresses are caught by the blacklists.

[8] We assume that updates are delivered within a minute of the time that they are generated.

[9] It is common for a spam-bot to send multiple e-mails to the same recipient consecutively.

[10] Microsoft had its own version called SenderID.

[11] To facilitate Trinity's deployment, migration to a full peer involves running a single daemon. The plug-in can easily check if the daemon is running and run in "normal mode". To run the peer on a different host, some additional configuration is required.