Scalable Centralized Bayesian Spam Mitigation with Bogofilter

Jeremy Blosser and David Josephsen – VHA, Inc.

ABSTRACT

Bayesian content filters gained popular acclaim when they were put forward in 2002 by Paul Graham as a potential long-term solution for the spam problem. They have since fallen from the limelight, however, due to perceived attack vulnerabilities inherent to all content-based filters as well as real and imagined vulnerabilities specific to Bayesian filters. It has also been assumed that Bayesian filters would be problematic to implement in centralized or large environments due to wordlist management issues. This paper revisits the effectiveness of Bayesian filters as a sustainable singular spam solution for mid- to large-sized environments through a real-world study of the deployment and operation of the Bogofilter Robinson-Fisher Bayesian classification utility in a production mail environment servicing thousands of accounts. Our implementation strategy and methodology as well as our results are described in detail so that they can be evaluated and replicated if desired. Other filtering methodologies which were previously implemented in this environment are also discussed for comparison purposes, though they have since been removed from production due primarily to lack of need. Bayesian classification has been able to solve the spam problem for this user population for the present and observable future, with a single wordlist, and with no secondary spam filtering techniques employed. Significantly, only two business-related legitimate messages have been reported as blocked due to filter misclassification since Bogofilter was deployed.

Introduction

Unsolicited bulk and commercial email, popularly known as spam, is one of the most critical issues facing systems, mail, and network administrators today. More and more human and system resources are being dedicated both to dealing with the day-to-day filtering of mail and to determining any possible long-term solutions to the problem, be they technical, social, or legislative. Whatever solutions are applied, however, are inevitably subverted or defeated by spammers within a short time of implementation, causing many to characterize the situation as an arms race. Fixed string content filters are avoided by mangling commonly blocked words. MTA blacklists cause spammers simply to change their mail routes and service providers and cause excessive collateral damage [JAC]. Challenge-response systems have led to spammers adding mail route harvesting to their existing address harvesting practices and cause similar collateral damage [SEL]. Message repositories and checksumming databases are brute force solutions with high false negative rates [MER]; in our experience they also require persistent high maintenance and ever-increasing resource utilization. The most recent attempts to add authentication into the mail delivery process through extending or replacing SMTP seem likely to be the most costly yet in terms of collateral damage and infrastructure costs [KNO], yet spammers are already observably capable of bypassing these measures using hijacked end-user machines to send messages using the local mail submission system and routing through authenticated channels, in the same way that email worms currently propagate [JdeBP]. This is not to say that these technical methods are entirely without merit or usefulness, as they are all effective in at least blocking some percentage of spam. However, the cost of these incremental cures is escalating to the point they may become as bad as the disease itself.

Legislative methods are in their infancy but are already being undermined by political pressures and seem likely primarily to force spammers to move even more of their operations to countries with friendlier laws, something they have demonstrated extreme willingness to do. Diligence on the part of prosecutors may yet produce results here, but they are likely to be a long time coming. The problem is here today, and its current severity can not be overstated. End users who are the worst affected are reporting hours spent per week deleting and attempting to block unwanted mail that is increasingly offensive in nature, and more and more are determining the effort of keeping their email usable is not worth it and are instead returning to other forms of communication.

When Paul Graham published his 2002 paper ‘‘A Plan for Spam’’ [GRA], which proposed applying Bayesian statistical modeling as a method of content filtering and provided very promising early results from Graham’s own tests, many in the anti-spam and end-user communities lauded the approach as a possible permanent solution. Bayesian filtering claimed all the advantages of content filtering, while adding learning algorithms to defeat message character changes over

time and token-mangling attacks. It also promised an extremely minimal percentage of legitimate mail incorrectly classified as spam (false positives), and added a level of personalization in block lists that was unprecedented. Multiple Bayesian filter implementations appeared virtually overnight from both hobbyist and commercial sources, ranging from open source projects such as Bayespam and Bogofilter to implementations in Mozilla Thunderbird, Microsoft Outlook, and Apple's Mail program. Bayesian filtering fell from the limelight nearly as quickly, however, due to predicted difficulties in large-scale or centralized implementations, some success by spammers in defeating early Bayesian implementations using poisoning attacks, assumptions of vulnerabilities common among other content filters, and other issues both real and imagined. Academics and pundits continue to discuss the value and potentials of Bayesian methods, and Bayesian classifiers are still deployed alongside other filtering systems, but a popular opinion among administrators and the public seems to be that Bayesian filters are no better at providing a long-term solution to the problem than other methods [BAA, ALL, EMM, PAG, WAR, JdeBP2, BOW].

These dismissals are demonstrably premature, however. We implemented Bogofilter in a centralized capacity using common wordlists for an environment with thousands of accounts in April of 2003. Despite the best efforts of spammers to defeat it, Bogofilter has continually exceeded all expectations as a filter and has effectively solved the spam problem in our environment, allowing us to remove all other spam-specific filters from our architecture. Though we initially only set out to bring the problem under control and not necessarily to eliminate all spam from our environment, our estimates indicate we are currently able to accurately block 98-99% of the spam sent to us. The filter blocks an average of 1,100 incoming spam messages per hour (700,000 to 1 million spam messages per month), or 60-75% of our incoming mail volume. This amounts to roughly 40 spams per user per work day. We have vastly exceeded management's goals as well as our own, and our users are able to conduct business without constant solicitations appearing in their inboxes. Perhaps most importantly, only two business-related legitimate messages have been reported as blocked by this filter since it was implemented more than a year ago. While no solution is likely to last forever, our results indicate it is possible that properly configured and deployed Bayesian filters are capable of sustaining a high enough success rate that administrators may finally be able to stop spending all their time refining filters and instead focus on securing end-user machines, thus finally moving to take the offensive in the spam war.

Environment and Goals

Company and Environment

Our company, VHA Inc., is an alliance of not-for- profit hospitals, health systems and their affiliates. Member organizations range from single 50-bed facilities to large, integrated health care systems made up of multiple hospitals, physician clinics, and support care sites. Notable member organizations include Baylor Health Care System, Cedar-Sinai Health System, Mayo Foundation, and Yale-New Haven Health Services Corp.

The mail environment in question provides mail services for more than 2,000 accounts distributed between our corporate headquarters and approximately 30 regional offices nationwide. Monthly mail volume over the past year averaged 1.85 million messages per month, 70% incoming and 30% outgoing. Based on filter logs and user feedback, we estimate today that on average 65-70% of the incoming mail (nearly 50% of the total mail volume) is spam. The content of our mail is typical of any company our size; the one exception is that since we operate in the health care industry and specifically the medical supply purchasing industry, we see quite a lot of legitimate mail having to do with pharmaceutical contracts and supplies, i.e., messages mentioning Viagra and other drugs can and do show up in our legitimate mail flow.

Architecture

All mail entering or leaving the environment passes through a centralized pair of load-balanced mail exchangers running the qmail MTA. The first exchanger currently has dual 1 GHz Pentium III's, the second has dual 700 MHz PIII's. Both have 1 GB RAM. Incoming mail is handed off to a Microsoft Exchange system which also handles all internal mail (Figure 1). Outgoing mail originating at the Exchange system is handed to the mail exchangers directly, while mail originating from internal applications uses a separate internal qmail server for relaying to the outside. All of our spam filtering efforts have been targeted at the qmail environment due to resource utilization issues and end-user interaction concerns.

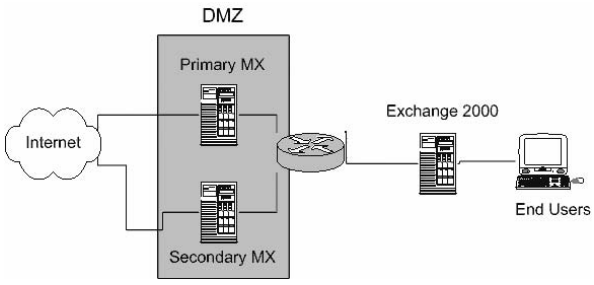

Figure 1: Mail environment architecture before Bogofilter.

The current Bogofilter implementation added one additional server to our environment. This server is responsible for caching mail as required for filtering purposes, receiving end-user filtering corrections, providing the environment that administrators use for processing corrections and ongoing training of Bogofilter, and holding the master copy of the wordlists (Figure 2). This server currently has dual 2 GHz Xeons and 2

GB RAM, but rarely has a CPU load average higher than 0.1. More information about this server is provided in the "Design" section below.

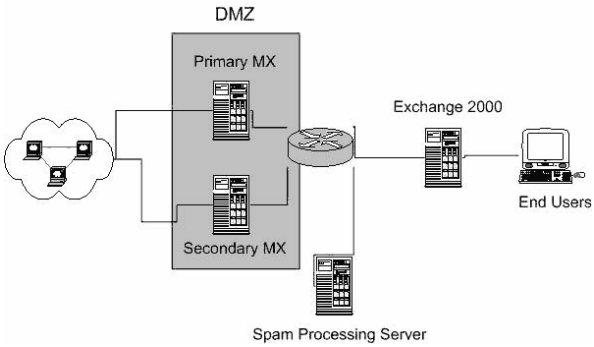

Figure 2: Mail environment architecture after Bogofilter.

Goals

Our goal was not to eliminate spam completely from our environment, but to bring the problem under control so that normal work could go on unaffected. The initial target was to block 75% of our spam, which we initially estimated would be 30% of our incoming mail volume. We also needed to guarantee that we would not block any legitimate business mail in the process of blocking spam. Given our mail volume, anything we implemented needed to be fast and efficient enough not to make excessive demands on our existing infrastructure. As a final goal, we wanted to keep the blocking centrally managed at the server level, rather than something that the users would have to deal with.

Initial Filtering Attempts

Basic Filters

Basic initial attempts at blocking via fixed-string matches against the sender address, reverse DNS lookups, and even a short-lived block of mail from all non-US domains met with predictable results. We quickly learned that the spammers we were dealing with were not just sending indiscriminately to global email lists, but were specifically monitoring the delivery status of spam entering our environment and were willing and able to change their messages to avoid our filters. We were not willing to spend hours each day creating new filters which would be made obsolete within a week, so we began to look in earnest for a more viable long-term solution.

Due to the nature of the spam arms race and our goal of efficiency we decided to target our efforts at automated content-based filtering. Blacklists were not considered an option due to the requirement to avoid blocking any form of legitimate mail. Challenge-response systems placed unacceptable burdens on our business contacts while simultaneously being considered too easy to spoof. More dramatic efforts aimed at user and mail route authentication seemed unlikely to provide any long-term relief, since we assumed they would primarily force spammers to continue their recent attacks on user computers themselves, using zombies and the local user's own credentials to send spam through valid mail routes. Spam is not spam without the content, however, and content-based filtering appeared to offer the most desirable combination of accuracy and tunability. Scalability was the most likely point of failure, but we hoped that a sufficiently automated system could handle the load.

Vipul's Razor

Vipul's Razor, commercially distributed through Cloudmark, was selected for a pilot. This is a checksumming content filter; each incoming mail is compared over the network in real time to a database of known spam messages. This database is fed by user submissions and spamtrap addresses maintained by Cloudmark. The advantages of this type of system are that it has a negligible false positive rate and is relatively difficult to attack directly. The primary working attack is to find a way to pollute the spam database or otherwise disrupt the checksum sharing process. The disadvantages include scalability and ongoing maintenance; someone, somewhere, must receive each new spam before it can be stored for future comparison.

While the implementation was effective in blocking 30% of our incoming mail volume as spam, it introduced significant system overhead and proved ultimately unscalable. This was primarily due to the extra network traffic required to contact the checksumming database and, even more significantly, the Perl instantiation required per message on the mail exchangers for the Razor client. Our environment experienced more service delays and outages than it ever had previously, due to both intermittent network errors in reaching the database servers and excessive CPU load produced by the clients.

Most important, however, was the fact that our end users reported no noticeable change in the volume of spam they were dealing with, despite the 30% reduction in overall mail volume. Complaints of offensive spam during this period actually increased. This was our first indication that our initial estimate of the size of our problem was off by a wide margin. We considered moving to a local checksum-based solution such as DCC, but eventually concluded that this method was too high maintenance to be viable in our environment. We began researching other alternatives, with even more attention paid to the real world system requirements imposed by prospective solutions.

Bogofilter Implementation

Bayesian Classification and Filtering

Although there was work being done on Bayesian classification of spam as early as 1996, Paul Graham's paper [GRA] is generally credited as being the first description of a solid implementation with

promising results. Gary Robinson later improved on Graham's work, suggesting algorithm modifications to make Graham's technique mathematically consistent with Bayesian statistics [ROB]. Robinson's probability-combination improvements make use of the inverse chi-square function first described by Ronald Fisher and are therefore sometimes referred to as the Robinson-Fisher algorithm.

Probability theory holds that the probability of a given event can be plausibly estimated by observing how often it has occurred in the past under similar circumstances. Bayesian probability begins by creating a "prior probability distribution," or "prior," which estimates the probability of given events. The prior can be used to calculate the likely outcome of new events. Then experiments are run and the outcomes recorded. The prior can then be updated to reflect the outcome of the experiments. Bayesian algorithms are self-correcting in this respect: they learn from their mistakes.

Token Scores

If an email is viewed as a collection of discrete words, or "tokens," a score can be assigned to each token. These scores are roughly comparable to probabilities in that they directly correspond to a token's "spamminess" or "non-spamminess." Each token's score represents how likely that token is to appear in an email composed of tokens that are uniformly distributed and statistically independent. The more "spammy" or "not spammy" a token is, the less likely it would be to appear in such an archetypal email. That is, the presence of these "interesting" tokens in an email violates statistically neutral linguistic behavior in measurable ways.

These scores are easy to calculate given a relatively even number of manually sorted spam and non-spam emails; the math is described in detail by Robinson in his paper "Spam Detection" [ROB]. The tokens themselves, plus the frequency with which they occurred in the text of spam and non-spam messages previously used to train the filter, are used to create a prior probability distribution in database form. Most Bayesian spam implementations are "objective" in that they put a lot of thought into assigning the prior, especially for tokens that do not currently exist as part of the prior. Bogofilter is no exception. When it encounters a new token, it assigns it the value of the variable `robx` ("Robinson's x"). `robx` is calculated as the average of the scores of all the other known tokens. The variable `robs` ("Robinson's s") is used in situations where Bogofilter has only seen the token a few times (low data situations). `robs` acts as a user-defined metric of trust and limits `robx`'s effect in low data situations.

Combined Scores

Once the prior is used to calculate a score for each word in an email, these token scores must be combined into a single score, which is representative of whether or not the given email is spam. Bogofilter uses the Robinson-Fisher algorithm for probability combination by default. There are three important aspects to the inverse chi-square driven algorithm Robinson has provided. First, it removes assumptions that were not relevant in the context of spam filtering, such as assuming the probability of a given token's prediction being correct is the same whether its outcome is spam or non-spam. Second, it is more sensitive to token scores that indicate if a message has an underlying tendency toward spam or not spam. Third, it uses a user-defined variable to decide how many "interesting tokens" to combine, which more consistently handles both large and small emails. For Bogofilter this variable is named `min_dev`. `min_dev` is the minimum deviation from the neutral score of 0.5 a token must have to be considered "interesting" and therefore be included in the classification of the message as a whole.

The other user-defined variables exist in the form of cutoff values to which the combined score of the email is compared. In binary classification, emails with scores over a singular threshold are considered to be spam. Those under it are not. Bogofilter optionally allows for three-factor classification. Two cutoffs are provided, `spam_cutoff` (for spam) and `ham_cutoff` (for non-spam). Anything in between is labeled "unsure." In practice, "unsure" mails provide interesting fodder for training, so this is the classification method we use.

Selection of Bogofilter

Graham's paper had been published for some time at this point, and like the rest of the industry, we were intrigued by his reported success. The concept of a Bayesian approach appealed to us, given our experience with shifting spam patterns and our desire to stick with content-based filters. The nature of the filtering, however, predicted the best results when each user maintained individual wordlists to most accurately reflect the unique nature of individual mail spools.[1] Our goals of scalability and minimal user intervention were at odds with this, but we decided to do some initial testing to see if it could work with a single set of wordlists for an entire organization.

We began monitoring the field of filters claiming a Bayesian implementation, and quickly settled on Bogofilter. Although other implementations had features Bogofilter at the time lacked, it was an obvious choice to us for its small overhead and its attempt to work within the Unix philosophy of "do one thing and do it well." Bogofilter is written in C and expects to operate on standard I/O streams, adding a custom header to messages it operates on and/or indicating message status with an exit code. This fit our environment perfectly. There are other Bayesian filtering

[1] Ironically, a primary impetus for the original creation of Bogofilter was to create a filter which implemented Graham's proposals but could be quickly deployed by individuals [ESR].

systems which tend to worry less about efficiency than dealing with cutting edge theory; these are ideal for experimental approaches and furthering the state of the art but are less useful in a high-volume production environment. This is not to say that Bogofilter's implementation is not correct, however; the Bogofilter development group expends a good deal of effort ensuring that their implementation of Bayesian algorithms is correct, as well as tracking changes and advances in Bayesian filtering theory and providing their own measurements and contributions to the field. Bogofilter strives to provide the best of both efficiency and accuracy and was therefore a good choice for our environment. Selection of this tool has no doubt contributed heavily to our filtering success.

Training

Another likely source of our success is our training process, both for initial wordlist seeding and subsequent training and corrections. Bayesian filtering is unfortunately not a turnkey-style solution; while it is possible to implement a Bayesian spam filter (including Bogofilter) by simply following the steps in a HOWTO and running some scripts, best results require that the administrators have some understanding of the theory and how best to apply it to their environment. This is primarily relevant in the initial and ongoing training process. We spent a fair amount of time gaining an understanding of the theory and the work being done to apply it to spam filtering and Bogofilter's specific implementation before moving to implement, and we designed our training process accordingly.

Sorting

Proper training requires a large pre-sorted collection of spam and non-spam messages. It is nearly impossible to create an effective Bayesian classifier using only a handful of mails. The filter needs to be trained on at least several thousand messages each of spam and non-spam from the start, preferably in nearly equal amounts [LOU].

To seed the initial wordlists we therefore collected several days' worth of incoming and outgoing mail at the mail exchangers. This provided us with more than 220,000 messages, which we then classified manually into groups of spam and non-spam. Approximately half of these were discarded as mailer daemon traffic such as bounces, delivery overhead, and virus quarantine messages. All outgoing mail was automatically classified as non-spam but was kept to provide a large body of known good mail so that if the filter established any biasing error it would be in favor of keeping legitimate mail. The remaining mail was classified in successive phases. Mails were manually sorted by one administrator while another looked for patterns in the already classified mail to allow for batch classifications to speed up the process. These batch filters were similar to the ones employed by other anti-spam software; while not long-term options for our environment, they worked in the short term due to the static nature of the mail snapshot we were operating against and the lower efficiency requirements. Anything faster than a human was a benefit here. Further, this processing was done in a development environment away from the regular mail exchangers, so regular mail load was not affected. We also took some time to develop interactive shell scripts to aid the manual classification process, both to speed it up and to preserve as much privacy as possible. The scripts provided only the headers of messages, color-coded to highlight suspicious patterns. The full messages were available at operator request, but were rarely needed in the initial classification stages.

After approximately 30,000 messages had been sorted, we began to incorporate Bogofilter itself as a sorting tool. We tested its effectiveness by running it against 20,000 pre-sorted messages, 10,000 each randomly taken from the sorted collections of spam and non-spam. Bogofilter without any established wordlists was presented with random individual messages from these collections and asked to determine if they were spam or non-spam. If its classification agreed with the human classification, no action was taken. If its classification disagreed with the human classification or Bogofilter was unsure of how to classify the message, it was corrected based on the human classification (this process is known as ''train on error''). The output of the classification versus the human classification was presented in real time during the test run to an administrator (Figure 3). At the end of the run, messages Bogofilter had consistently gotten wrong were further investigated.

Results at this early stage were simply shocking. At the start of the run Bogofilter would tend to get a few messages wrong until it had a handful of each type of message in its wordlists, at which point it would immediately begin to get the vast majority of classifications correct, increasing dramatically and observably until it had seen several thousand messages, at which point it would generally stabilize at around 95% accuracy. Inaccuracies were either false negatives or ''unsures'' in all but a handful of cases. More often than not messages which Bogofilter persistently had trouble classifying were re-evaluated to find that the human administrator had gotten them wrong in the first place and Bogofilter was correcting us.

Once we had performed several of these test runs we were confident Bogofilter could be utilized as a sorting tool. We created wordlists based on all the messages sorted so far and then ran Bogofilter against the remaining unsorted lists, allowing it to sort them into provisional groups. These groups were then further evaluated manually by an administrator to verify the accuracy of their sorting before they were added to the global collections.

Eventually, nearly 20,000 messages were removed due to excessive ambiguity about their nature.

These included news updates, potentially legitimate commercial mailings, religious and inspirational messages, joke-of-the-day lists, and others. We decided it would be better to move forward with these messages unclassified and allow users to give more feedback during the pilot phase of the process.

This sorting process took both administrators the better part of a week to complete, and we became intimately familiar with the character of the spam we were receiving. This was very tedious work, but there is no doubt taking the time to do this was a key to the current success of our implementation.

Configuration and Tuning

Once the initial message sorting is complete, the filter must be configured and tuned for its environment. There are several variables to consider, along with their interaction, and we attempted to use appropriate values for each during each stage of the training process. On some occasions we did test runs using various combinations of settings to see which provided the best results. The theories involved were still being formed and actively debated, so we were experimenting pragmatically to find the best settings, but in most cases the theoretical and empirical work that has been published since agrees with our results. On a related note, even though Bogofilter at the time shipped with tools to perform all of this training and tuning, these proved too nascent and incomplete to meet our needs, so we developed our own. Current versions of Bogofilter ship with complete training tools which provide even more rigorous tuning functionality than what is described here. We are also deeply indebted to Greg Louis for his excellent tuning documentation [LOU].

```
% head -n 25 t.log | sed -f t.sed
        NOTSPAM/1051176904.31414.XXXXXXXX: NOTSPAM:  Legit:   spamicity=0.00e+00
            SPAM/1051295846.6703.XXXXXXXX:    SPAM:   Spam:   spamicity=1.00e-00
        NOTSPAM/1051277931.15769.XXXXXXXX: NOTSPAM:  Legit:   spamicity=0.00e+00
        NOTSPAM/1051198384.24223.XXXXXXXX: NOTSPAM:  Legit:   spamicity=7.37e-04
            SPAM/1051151402.7668.XXXXXXXX:    SPAM:  Legit:   spamicity=8.77e-03
        NOTSPAM/1051316490.31430.XXXXXXXX: NOTSPAM: Unsure:   spamicity=0.476275
        NOTSPAM/1051212317.20791.XXXXXXXX: NOTSPAM:  Legit:   spamicity=7.00e-06
        NOTSPAM/1051198083.18275.XXXXXXXX: NOTSPAM:  Legit:   spamicity=1.66e-04
           SPAM/1051226341.24583.XXXXXXXX:    SPAM: Unsure:   spamicity=0.499488
        NOTSPAM/1051160935.17463.XXXXXXXX: NOTSPAM:  Legit:   spamicity=2.62e-03
        NOTSPAM/1051300472.26807.XXXXXXXX: NOTSPAM:  Legit:   spamicity=3.63e-03
           SPAM/1051307951.28054.XXXXXXXX:    SPAM: Unsure:   spamicity=0.500000
           SPAM/1051268971.18425.XXXXXXXX:    SPAM:   Spam:   spamicity=1.00e-00
        NOTSPAM/1051223545.17452.XXXXXXXX: NOTSPAM: Unsure:   spamicity=0.500000
           SPAM/1051187116.12926.XXXXXXXX:    SPAM: Unsure:   spamicity=0.500210
            SPAM/1051176998.2277.XXXXXXXX:    SPAM: Unsure:   spamicity=0.499942
        NOTSPAM/1051228484.31637.XXXXXXXX: NOTSPAM:  Legit:   spamicity=2.13e-14
        NOTSPAM/1051185896.25758.XXXXXXXX: NOTSPAM: Unsure:   spamicity=0.455169
           SPAM/1051246738.10352.XXXXXXXX:    SPAM: Unsure:   spamicity=0.500008
            SPAM/1051280016.3280.XXXXXXXX:    SPAM: Unsure:   spamicity=0.503970
        NOTSPAM/1051199776.12275.XXXXXXXX: NOTSPAM:  Legit:   spamicity=5.12e-05
           SPAM/1051284997.14278.XXXXXXXX:    SPAM: Unsure:   spamicity=0.572632
           SPAM/1051302521.23943.XXXXXXXX:    SPAM:   Spam:   spamicity=1.00e-00
        NOTSPAM/1051284875.12731.XXXXXXXX: NOTSPAM: Unsure:   spamicity=0.303452
        NOTSPAM/1051220593.13157.XXXXXXXX: NOTSPAM: Unsure:   spamicity=0.394604
%
% tail -n 25 t.log | sed -f t.sed
        NOTSPAM/1051210780.27932.XXXXXXXX: NOTSPAM: Unsure:   spamicity=0.129265
           SPAM/1051175772.21302.XXXXXXXX:    SPAM: Unsure:   spamicity=0.663503
            SPAM/1051257860.3202.XXXXXXXX:    SPAM:   Spam:   spamicity=9.99e-01
        NOTSPAM/1051193288.23220.XXXXXXXX: NOTSPAM:  Legit:   spamicity=1.06e-10
        NOTSPAM/1051193593.21449.XXXXXXXX: NOTSPAM:  Legit:   spamicity=0.00e+00
            SPAM/1051136802.3620.XXXXXXXX:    SPAM:   Spam:   spamicity=9.98e-01
        NOTSPAM/1051208078.15694.XXXXXXXX: NOTSPAM:  Legit:   spamicity=7.11e-03
        NOTSPAM/1051138154.23411.XXXXXXXX: NOTSPAM:  Legit:   spamicity=0.00e+00
           SPAM/1051175724.22780.XXXXXXXX:    SPAM:   Spam:   spamicity=9.59e-01
         NOTSPAM/1051276954.4253.XXXXXXXX: NOTSPAM:  Legit:   spamicity=9.08e-12
           SPAM/1051266551.15139.XXXXXXXX:    SPAM:   Spam:   spamicity=9.75e-01
          NOTSPAM/1051259750.482.XXXXXXXX: NOTSPAM:  Legit:   spamicity=8.68e-04
           SPAM/1051252160.15717.XXXXXXXX:    SPAM:   Spam:   spamicity=1.00e-00
            SPAM/1051225816.5567.XXXXXXXX:    SPAM:   Spam:   spamicity=9.84e-01
           SPAM/1051179746.27924.XXXXXXXX:    SPAM:   Spam:   spamicity=1.00e-00
           SPAM/1051268772.31580.XXXXXXXX:    SPAM:   Spam:   spamicity=1.00e-00
        NOTSPAM/1051301801.15206.XXXXXXXX: NOTSPAM:  Legit:   spamicity=2.79e-09
        NOTSPAM/1051191245.27903.XXXXXXXX: NOTSPAM:  Legit:   spamicity=1.42e-13
           SPAM/1051205684.30817.XXXXXXXX:    SPAM:   Spam:   spamicity=1.00e-00
           SPAM/1051287773.16831.XXXXXXXX:    SPAM:   Spam:   spamicity=9.91e-01
            SPAM/1051316745.1425.XXXXXXXX:    SPAM:   Spam:   spamicity=9.69e-01
         NOTSPAM/1051203657.6582.XXXXXXXX: NOTSPAM:  Legit:   spamicity=8.39e-09
           SPAM/1051233957.9233.XXXXXXXX:    SPAM: Unsure:   spamicity=0.830934
           SPAM/1051291673.32736.XXXXXXXX:    SPAM:   Spam:   spamicity=9.59e-01
           SPAM/1051251862.11339.XXXXXXXX:    SPAM:   Spam:   spamicity=9.99e-01
%
```

Figure 3: Screenshot of the beginning and end of a preliminary Bogofilter classification run of our sample mail corpora.

As noted above, Bogofilter's primary configuration variables in tri-state classification mode are `robs`, `robx`, `min_dev`, `spam_cutoff`, and `ham_cutoff`. `spam_cutoff` and `ham_cutoff` default to 0.95 and 0.10, respectively. For the sorting process above we reduced `ham_cutoff` to 0.05 to require a higher level of certainty from Bogofilter before it was allowed to flag a message for the legitimate corpus. For the training process we also used 0.05 to inflate the number of legitimate mails that would be classified as ''unsure'' and therefore used for training during the ''train on error'' process. These were more safeguards to ensure any biases introduced during training would be on the side of blocking too little mail instead of too much.

To determine appropriate values for `spam_cutoff` and `robs`, we ran several iterations of a training script using a methodology similar to recommendations published by Greg Louis [LOU2, LOU3]. We divided our pre-sorted message collections into three randomized groupings. Group A had 20,000 messages (10,000 each of spam and non-spam). Group B had 10,000 messages (5,000 each of spam and non-spam). Group C had all remaining messages (approximately 45,000). Bogofilter was fully trained on each message in Group A, then trained on error for each message in Group C. Group B was then used as a test corpus, with no training done and errors tabulated. The train-on-error and test runs were repeated once. Following this, Group B was used to further train on error, then again used to test; this process was also repeated once. In this fashion Bogofilter was either fully trained or twice trained on error for every message in our corpora. See Appendix A for more detail.

We needed to determine if we should use the default `spam_cutoff` of 0.95 or if we could safely use a more aggressive value of 0.90. We also needed to determine if we should use a `robs` value of 0.01 or a more conservative 0.001.[2] We therefore ran the above test suite for each of the four permutations of these two values. While each of the four gave nearly identical results by the final two test rounds, the 0.01 and 0.90 combination had the highest accuracy during the initial rounds and was therefore selected (Figure 4). It was a pleasant surprise that using a lower `spam_cutoff` actually improved accuracy; apparently a statistically significant amount of our spam scored between 0.90 and 0.95, and using the lower cutoff meant more of these were classified correctly on the first pass.

For `robx`, we again cleared our wordlists and ran several iterations of a similar training script, beginning with fully training on 10,000 each of spam and non-spam, then training on error for the rest of the messages. At the end of each iteration we took the final `robx` value and used it as the initial `robx` value for the next iteration. We did this several times, until the `robx` value stabilized at 0.477112.

The final value is `min_dev`. We left this at the default of 0.1 throughout training and into production. The Bogofilter tuning documentation recommends moving this to somewhere between 0.3 and 0.46 to take into account fewer words per message, but testing

[2] At the time there was some debate about whether overly conservative `robs` values might cause unpredictable results. Louis has since conclusively determined that values lower than 0.01 do cause problems if a token occurs a few times in one wordlist but not at all in the other [LOU4].

robs/spam_cutoff	round	fpos	fneg	uspam	unotspam	err	percent
0.001/0.95	1	3	5	104	64	176	1.76%
	2	3	5	107	63	178	1.78%
	3	0	0	14	10	24	0.24%
	4	0	0	10	5	15	0.15%
0.010/0.95	1	3	7	87	47	144	1.44%
	2	3	8	83	48	142	1.42%
	3	0	0	15	11	26	0.26%
	4	0	0	10	4	14	0.14%
0.001/0.90	1	3	5	88	65	161	1.61%
	2	3	5	90	63	161	1.61%
	3	0	0	11	11	22	0.22%
	4	0	0	10	6	16	0.16%
0.010/0.90	1	3	8	72	46	129	1.29%
	2	3	8	70	47	128	1.28%
	3	0	0	11	11	22	0.22%
	4	0	0	10	5	15	0.15%

```
fpos    : false positives
fneg    : false negatives
uspam   : unsure (spam)
unotspam: unsure (not spam)
err     : total errors
percent : percent error
```

Figure 4: Results of varying the value of `robs`and `spam_cutoff`.

on our part showed that this reduced accuracy, and recent experiments with raising the value to 0.3 resulted in an immediate and dramatic increase in the amount of spam that made it through the filter. However, we will likely need to revisit this as wordlist attacks become more focused.

At this point the tuning was considered complete. Apart from raising the `ham_cutoff` back to 0.10 (to avoid an excessive number of ''unsure'' classifications), these were the wordlists and configuration we took to production (there are other options such as how the lexer should tokenize HTML tags and IP information; we have left these at the defaults). Throughout the tuning process we remained amazed at the speed with which Bogofilter gained accuracy during the training runs, and concerns that using a single set of wordlists for an organization of this size would lead to significant misclassifications appeared unfounded. Although we tried to keep in mind that our results were preliminary, saying that we had significant remaining doubts by the end of the training iterations would be a mischaracterization.

Design

As noted above, our border mail architecture consists of a pair of mail exchangers running qmail. Prior filtering attempts ran as mirrored installs on both of these servers. Bogofilter, however, required some method of maintaining the same word list on both servers and any future servers added for expansion purposes. We also needed a central location to receive end-user misclassification notices so that an administrator could process them and train Bogofilter on filtering errors. We knew this central host would likely need to be able to perform extensive training and other processing operations which we would not want to impact our general mail load. None of this processing, however, needed to happen in real time for the mail to be properly filtered. We therefore added a new server to provide for general spam processing. When filter corrections are necessary, the end user forwards the misclassified message to this server, where an administrator verifies the request and trains Bogofilter, updating the central copy of the wordlists which are stored on this server. These wordlists are updated on the mail exchangers nightly. Since the mail exchangers have their own copies of the wordlists, there is no effect to mail flow if this server is down or otherwise unreachable. Finally, the spam server maintains data on all the training operations that have occurred so that the wordlists can be recreated from scratch as required.

The actual filtering happens at the mail exchangers during the SMTP exchange. We use the qmail-qfilter wrapper around qmail-queue to provide real-time message filtering options. Our qfilter invocation first notes the mail in the filtering logs, then pipes the message through Bogofilter, then copies the mail to the spam server cache, then checks the Bogofilter-generated spamicity header to determine whether or not the message is spam (Bogofilter is also able to indicate message status with an exit code, but passthrough mode and header-based filtering proved more compatible with the qfilter and spam caching server implementation). If the message is spam, qfilter exits 31, which causes qmail to refuse to accept the message from the originating server. See Appendix B.

The solution of refusing to accept spam mail during the SMTP exchange has become somewhat popular because it dodges the problems of network congestion and server load created by attempting to send bounce messages to senders that do not exist. We also chose it because in the case of false positives it lets the legitimate sender know immediately that their message was refused, allowing them to follow up quickly with their intended recipient. External relay servers may in theory still create unnecessary bounces or cause delays in receipt of legitimate bounces, but this was determined the best fit for our environment. However, if spammers begin to train ''evil'' Bayesian filters to attack ''good'' Bayesian filters (as John Graham-Cumming has suggested [JGC]), we will need to reconsider any implementation aspects such as this one which indicate to spammers the real delivery status of their messages.

To create our ongoing training framework we augmented the scripts we had written for the initial wordlist creation. Mails users submit for correction are viewed on the spam server by the administrators either interactively in Mutt or using a custom script that iterates across the entire queue. In either case, the administrator is presented with the mail headers and information on how Bogofilter classified the message initially and how it is currently classified (Figure 5). In some cases Bogofilter will have changed the way it classifies a given mail based on prior corrections, and the message can be skipped. If Bogofilter still misclassifies the message, the administrator can view the message headers and body and provide correction as required. The training has been deliberately kept as a manual process to prevent user error from corrupting the wordlists. We also do not use Bogofilter's auto-update (-u) switch, as this is likely to introduce a significant amount of error in a high-volume environment.

While testing this configuration, the primary issue that had to be resolved related to the fact that the users were forwarding messages for correction after Exchange had delivered them. Exchange modifies the message headers (and in some cases the body) significantly, which means that the messages users forward us are not the same messages Bogofilter originally classified and are useless for training. The best solution to this seemed to be to cache the mails as Bogofilter originally saw them and look them up as required. The mail exchangers therefore forward a copy of all incoming messages to the spam server, where they are cached for a period of two weeks. The training lookup

scripts take the mails users forward, extract several pieces of header data, attempt to reconstruct the original headers, and look up the original message in the cache using these headers. Message headers are cached parallel to the full bodies to speed up lookups. This is quite frankly the least elegant piece of this entire filtering system, but no other option has been discovered for easily dealing with Exchange's header-mangling problem.

See also Greg Louis' article "Is Bogofilter Scalable?" [LOU5], which was written at approximately the same time as we were developing this implementation, and discusses many of the same issues. While Louis ends with uncertainty that the implementation he describes would scale enough to be able to handle an environment 10 or 50 times the one he describes (3,500 or 17,500 spam messages per day, respectively), our implementation demonstrates that a similar setup can scale to at least 75 times the spam volume he describes. We are not verifying all of our classifications as rigorously as he suggests, but so far this has not created a problem.

Implementation

The Bogofilter roll out to production was to occur in four phases. Phase 0 was a two-week trial by the CIO for his mail only, primarily so that he could verify no mail would actually be blocked until further testing had been done. Phase 1 was a one-month user-interactive pilot with the entire MIS department (approximately 80 users) participating. Phase 2 was a three-month user-interactive period with the entire company asked to participate. Phase 3 implemented Bogofilter in a decision-making capacity such that it began actively preventing the delivery of messages it classified as spam.

For Phases 0, 1, and 2, Bogofilter was implemented in passthrough mode alongside the existing Vipul's Razor deployment. Any mails that Vipul's did not block were filtered through Bogofilter, and a header was added with a classification for the "spamicity" of the message in numeric form. Outlook rules were created and distributed to the pilot groups to sort mails classified as spam and unsure into their own folders. Users were asked to review the contents of those folders on a periodic basis. Any legitimate mails found in the spam or unsure folders were to be forwarded to one address, and any spam messages found in the inbox or unsure folders were to be forwarded to another address. Administrators monitored those addresses and corrected Bogofilter on mails that had been classified incorrectly as described in the "Design" section above.

Some small training complications resulted from our attempt to maintain a single, company-wide set of

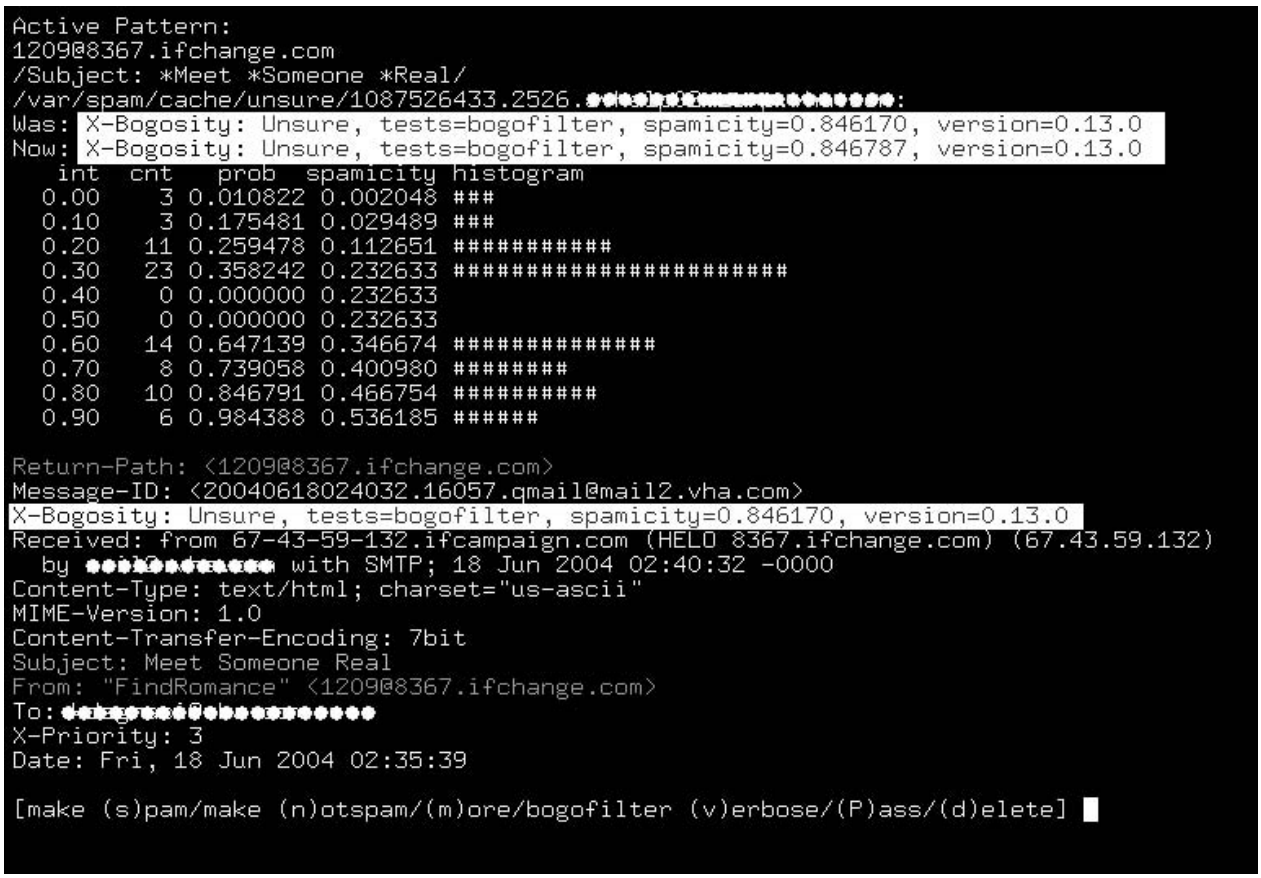

Figure 5: Screenshot of a spam message being displayed in our training tool.

wordlists. In reality, there is a large grey area between "spam" and "non-spam" with a user population of this size. "Joke of the day" lists, airline/hotel advertising, and religious messages are some examples. Worse, end-user collisions were initially frequent, where two users found the same piece of bulk mail in their unsure folders and one would report it as spam while the other reported it as non-spam. In an extreme case one user reported mail from the same home-improvement list as either spam or not spam, based on the home-improvement advice in question (roofing advice was spam, gardening advice was not). These were generally resolved by the administrators on a case-by-case basis, usually by either letting mob-rule prevail or allowing Bogofilter to decide based on the content of the mail in question. In the case of the home-improvement list, the administrators did as the user instructed, and Bogofilter adjusted accordingly.

Despite these minor complications, the pilot phases were an unqualified success. All but a handful of users reported all the spam gone from their inbox into either the spam or unsure folders. Though an implementation goal was to avoid user interaction and the final production system primarily does that, users seemed to appreciate the brief opportunity to do something about the spam that was reaching their inboxes. Most importantly, users were able to observe the filter in action and confirm first-hand that legitimate mail was not being misclassified; even during the pilot phases, no legitimate business-related mail was reported as misclassified by the filter. Finally, the administrators were confident at the conclusion of the pilot period that the maintenance and ongoing training framework was scalable and practical for the task at hand.

Once all parties were satisfied with the accuracy of the solution, the mail exchangers were reconfigured to use Bogofilter's spam classifications to block mail.

Results and Observations

Since its deployment in a blocking capacity, Bogofilter has continued to provide unprecedented accuracy and efficiency in mail filtering. Since the beginning of our spam mitigation effort, incoming email volume has risen from 900,000 to 1.4 million messages per month. Bogofilter has consistently

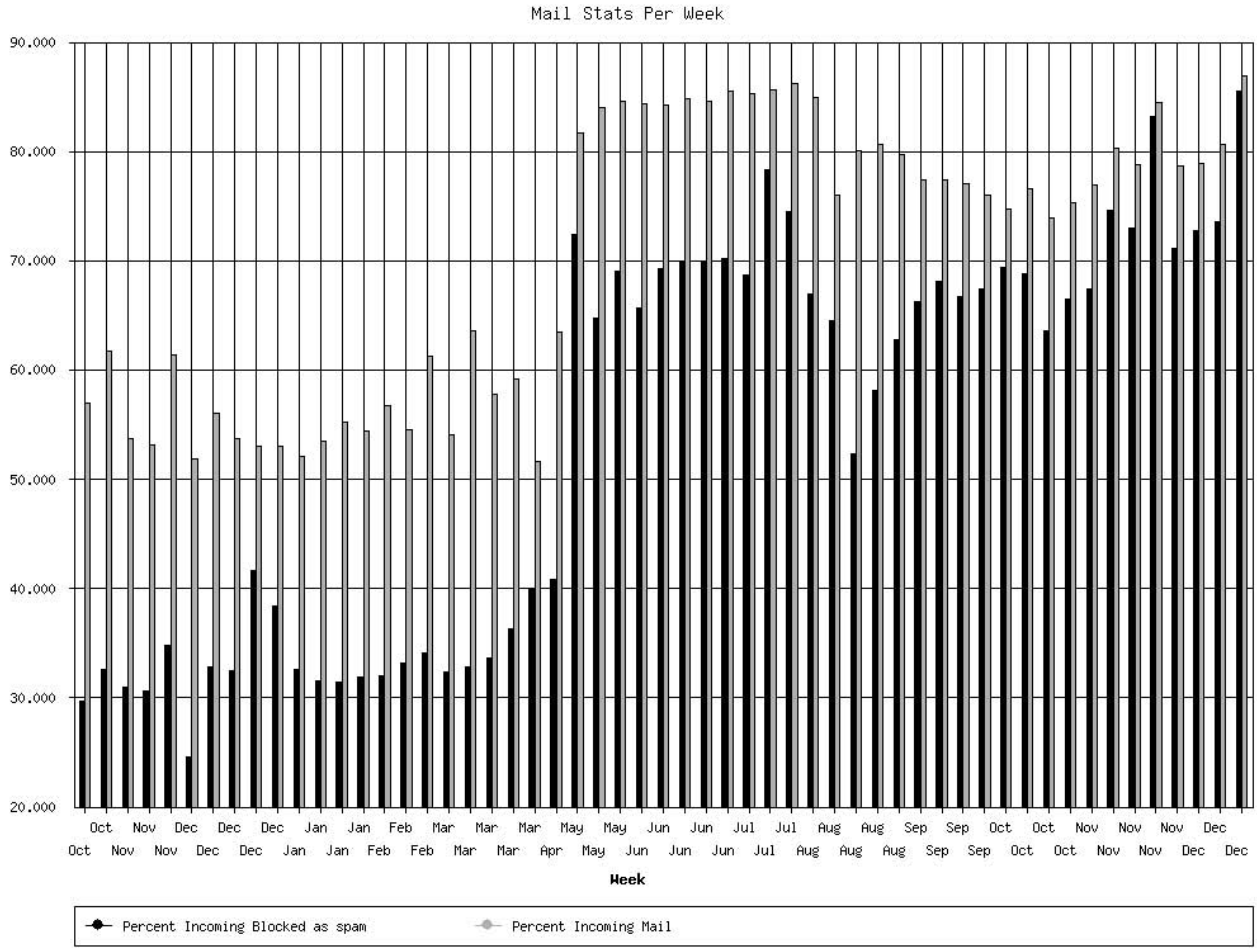

Figure 6: Percentage of inbound mail/Percentage of inbound mail blocked as spam, October 2002-October 2003. Bogofilter was installed in late April 2003. As we refused delivery of spam, we bounced less mail. As we bounced less mail, the outbound percentage dropped, and inbound percentages increased accordingly.

blocked 60-75% of this incoming mail as spam, or between 700,000 and 1 million messages on average per month. The filter averages 1,100 blocked mails per hour at the exchangers; when the average number of recipients per blocked mail is factored in, this is roughly 40 spams per user per work day. While it has been difficult to quantify exactly how much spam still enters our environment, based on end-user reports, mail logs, and manual inspection we believe the filter is 98-99% effective. This performance has persisted without fail for more than a year (Figures 6 and 7).

Further, in that time there have been only five blocks reported as false positives, only two of which were legitimate filter misclassifications and business related. One of these false positives involved a user mistakenly reporting legitimate mail as spam, then having similar followup messages blocked as spam. Another involved a group of users reporting a group of related mails as spam, then having a new user attempt to receive mail from that same source and having it blocked. A third was a user who had personal mail regarding a PayPal transaction blocked; as personal PayPal mails are not a common occurrence at our company, most of Bogofilter's opinion of PayPal tokens had been derived from spam categorization of PayPal phishing scam mails. All of these were resolved through either correcting the previous misclassifications or training on the blocked mail.

The two remaining misclassified messages were real false positives. The first was sent from a vendor providing an employee reward program. Though this was a legitimate business mail, its content was virtually indistinguishable from spam (e.g., "someone has sent you $xx.xx, click here to redeem it"). The other false positive was a bulk employee satisfaction survey sent from an external source. Some recipients received this mail fine, while others had it blocked. Inspection revealed that for some reason some of the mails had been sent with both Spanish and English language parts, while the rest were English only. The English mails got through, while the Spanish mails were blocked because those tokens had previously been encountered primarily in spam messages.

Since going to production we have not experienced any of the predicted downsides of using a single set of wordlists with centralized administration. It is possible that our user population's legitimate mail is more homogeneous than that of an average organization, but this seems unlikely. Rather, expectations that the filter's accuracy would "fall apart" when the same wordlists were used for many users seem simply not to have been borne out in practice. While even a success rate of 99% is an order of magnitude lower than the 99.9% and higher rates individual users of Bayesian filters have achieved, this does not necessarily indicate an inherent loss of discrimination ability due to shared

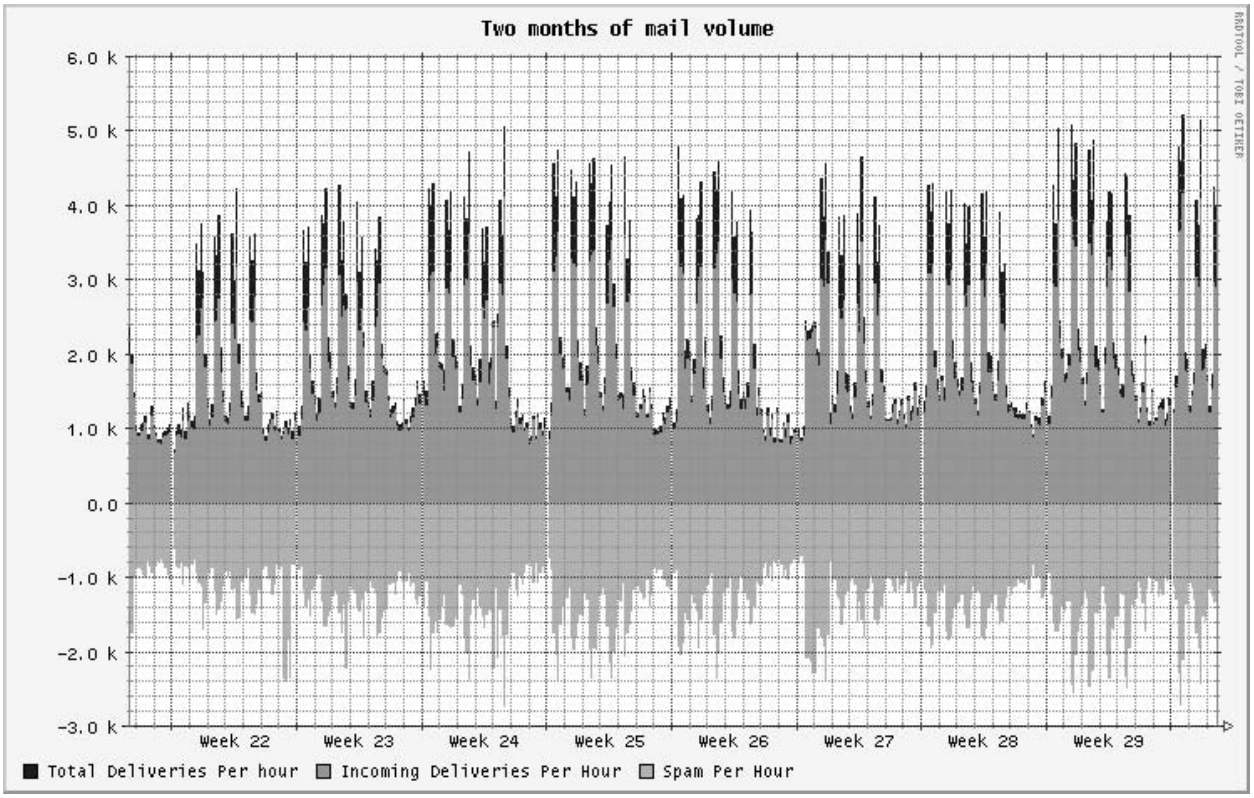

Figure 7: Overall mail traffic, August 2003-April 2004. Spam is multiplied by negative 1, to make the graph more readable. Data gaps are caused by log rotations.

wordlists. It could instead be the result of only training on the filtering errors users report instead of on all errors. In any case, blocking 99% of our spam is more than sufficient to meet the needs of an organization such as ours, and much better than any other solution we have evaluated or considered. The other minor wordlist-sharing issues encountered during the pilot of some users reporting mail as spam and other users reporting the same mail as non-spam seem to have resolved themselves; most likely those mails are accurately classified as spam, and the users who reportedly wanted to receive them are not actually noticing when they are blocked.

The effect of our blocking success on the end-user population has been dramatic and overwhelmingly positive, and the spam problem is solved from their perspective. Business has been able to continue without significant interruption. The only user complaints involved the minor number of misclassified mails referenced above. One of the more obvious signs of our success is that many of our users once again consider it odd to see more than a few spams a week in their inboxes, and will even call the support desk to complain when this happens. They continue to appreciate their ability to take some control of the problem when spams do reach them, and many speak of the filter in terms that make it obvious they are able to observe – and have even come to expect – nearly immediate results when they report new spams that have started getting into their inboxes and Bogofilter is trained on them. Periodically we are even contacted by other companies who have received word-of-mouth reports of our success and are interested in the details of our implementation; our users are apparently satisfied enough to mention our success when their contacts complain about spam.

Our ongoing training framework has also worked much better than we anticipated. In theory we should be monitoring all mails that are classified as "unsure" and using these to train Bogofilter. This would be a significant amount of effort, however, so we have instead limited our ongoing training to spam messages users report. Usually several users will report the same message as spam. Training on one message will generally be sufficient to update Bogofilter's opinion of duplicate or similar messages, so the training scripts re-examine each message before it is presented to the administrator. If Bogofilter correctly classifies the message on re-examination it is automatically skipped. On average we only need to train on 15-20 unique messages per day before Bogofilter has "caught up" and the rest of the training queue can be skipped. Using this methodology one administrator is able to process all the incoming requests in four hours per week. The continued success of the filter indicates that at least for the time being this training is sufficient.

So far this solution has also proved entirely scalable, with no obvious ceiling in sight. Mail exchangers can easily be added to support growing mail volume, with each only needing a Bogofilter installation, a copy of the configuration files, and the ability to pull the most recent copy of the wordlists from the spam server once per day. The system meets the goal of being efficient enough to operate in real time without affecting our mail-flow capacity; real-time Bogofilter evaluation of each message costs almost nothing and has not noticeably affected the system load or message processing time (Figure 8). The initial wordlists were 31 MB combined; that has only grown to 40 MB. The spam server is a potential bottleneck both in the need to receive a copy of every incoming message and the ability to lookup cached messages for training in real time, but the single server in place today maintains a CPU load average of 0.1, so this is not any kind of immediate concern. If message lookups become a

```
% ls -s -h random_words.txt
 12M random_words.txt

% < random_words.txt rl | head -c 10240 > words_10K.txt
% < random_words.txt rl | head -c 102400 > words_100K.txt
% < random_words.txt rl | head -c 1024000 > words_1000K.txt
% < random_words.txt rl | head -c 10240000 > words_10000K.txt

% /usr/bin/time sh -c '< words_10K.txt bogofilter'
0.06user 0.03system 0:00.08elapsed 101%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (496major+291minor)pagefaults 0swaps

% /usr/bin/time sh -c '< words_100K.txt bogofilter'
0.36user 0.03system 0:00.38elapsed 101%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (498major+440minor)pagefaults 0swaps

% /usr/bin/time sh -c '< words_1000K.txt bogofilter'
1.51user 0.10system 0:02.52elapsed 63%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (497major+978minor)pagefaults 0swaps

% /usr/bin/time sh -c '< words_10000K.txt bogofilter'
5.74user 0.14system 0:05.88elapsed 99%CPU (0avgtext+0avgdata 0maxresident)k
0inputs+0outputs (499major+1033minor)pagefaults 0swaps
```

Figure 8: Run times of Bogofilter for various message sizes, using our production wordlists. These times are relative to a dual-CPU 700MHz PIII, the least-powered server in this architecture.

bottleneck we can easily convert the header lookup to a faster database format or something similar. Extremely large organizations may find keeping up with the ongoing training is difficult, as four hours for this environment is potentially a full-time position for an organization with 10 times our mail volume. This can be further automated, however, using methods similar to those we describe in ''Future Work'' below.

As with any other filtering solution, we have observed spammers attempting to adjust their techniques to get around this filter. However, we have not seen them succeed, as evidenced by our ability to sustain a 98-99% block rate for more than a year. Initial attempts based on weaknesses and bugs in particular early implementations of Bayesian filters ceased to be effective long ago. Wordlist poisoning, token obfuscation, token dilution, and microspam attacks abound, but have had no significant effect on our ability to detect spam messages. This is despite a popular belief that they collectively represent an Achilles' heel of Bayesian filters [EMM, JdeBP2]. As Graham originally predicted, and he and others have since repeated [GRA2, JGC2], the attackers either completely fail to understand the nature of the system they are attacking, or the Bayesian filters simply adapt. Where wordlist attacks have had any success at all it is probable that Effective Size Factor and Bayesian Noise Reduction techniques will be effective in mitigating their value, as has already been demonstrated elsewhere [ROB2, ZDZ]. Microspams, which contain only a URL and a handful of words, show the most current potential for giving the filter trouble; however, each variant works at most once, since the URLs and the unique message headers immediately become high spam indicators, and the lack of other information provides no opportunity for non-spam tokens which could lower the spamicity of the message. In any case this is at best a temporary advantage for spammers, as there are filter enhancements to deal with these as well; some of these are discussed in the next section.

Future Work

Although this implementation is functioning well beyond expectations and has shown no degradation of performance in the time it has been running, there is room for improvement, both to increase filtering efficiency and to stay ahead in the arms race. The point of adaptive filters of course is that they will automatically keep up as spam changes, but there are process efficiencies and theory improvements to incorporate. There are also possible side-channel attacks we need to be wary of.

First, we are planning an upgrade to a more recent version of Bogofilter. We implemented on version 0.13.0. While this has proved stable and reliable in our configuration, it is rather out of date. Newer versions promise increased functionality and configuration options. In addition there are lexer changes to support continuing work in the field of applying Bayesian filtering to spam, such as the way HTML messages are handled. Most importantly, new versions of Bogofilter contain support for Effective Size Factor (ESF), which is designed to account for loss of filter discrimination introduced by naturally occurring volatility in the data as a result of token redundancy.[3]

In addition to upgrading Bogofilter, we would like to track the shifting nature of our spam more rigorously by being more proactive in training on messages which are misclassified or classified as ''unsure.'' As noted above, we currently primarily rely on end users to forward us misclassifications; while this currently works, at the moment it is only obvious to end users when mails that should have been blocked arrive in their inboxes. It is not obvious to them when legitimate mails on certain topics are flagged as ''unsure.'' As the wordlists are populated with more and more tokens that are only spam, we run the risk of introducing false positives.

One proposed method of addressing this issue is to make it visually obvious to end users when messages are classified as ''unsure'' so that they can forward them to the appropriate training address as they see fit. We do not want to inconvenience them by asking them to install rules to filter mails into separate folders again, so we are investigating ways to modify the way the message displays in Outlook. One option is to use the ''X-Message-Flag'' header to provide a message indicating the mail is suspected to be spam; Outlook will present this to the user as an alert header above the message body. If users cooperate, this would allow us to train on a higher percentage of unsure mails (especially those

[3] Spammy tokens tend to appear in groups. For example, an email containing the token ''mortgage'' is more likely to contain the token ''mortgage'' again, or other related spammy tokens such as ''refinance.'' This token redundancy can have a negative impact on filter discrimination by artificially magnifying the combined probability disproportionately between large and small emails. The good news is that problems with token redundancy can be dealt with mathematically; Gary Robinson has evidence to suggest that applying an Effective Size Factor (ESF) improves filter discrimination significantly by reducing the impact of this redundancy [ROB2]. Greg Louis' work confirms this [LOU6].

Note that token redundancy should not be confused with basic statistical independence, though the issues may appear similar. Confusion about this distinction has led to much ado about the violation of statistical independence in the context of Bayesian mail filtering because current approaches assume statistical independence of the input data where it does not actually exist. For example, training on error violates statistical independence by selecting messages for training based on prior examination, while the algorithms assume that training is done using a truly random sample of messages. Some have claimed this lack of actual independence represents a weakness in the use of Bayesian classifiers that is somehow ''exploitable'' [ALL]. Violations of assumptions of statistical independence are not news to probability theorists, however, since these assumptions are nearly always violated in practice [ROB3]. In addition, there is work to suggest that even the existence of the assumption of statistical independence has been overstated in the context of Bayesian classifiers [DOM].

which are non-spam) without requiring administrators to do the sorting manually.

However, we would prefer to rely on end-user interaction as little as possible. We intend to implement tools that will allow us to test Bogofilter automatically on incoming mails when we can make reasonable guesses on the status of those messages independently. Probable spam or non-spam messages which Bogofilter apparently misclassifies can be flagged for administrator review, and the filter can be corrected if necessary. Any spam-blocking methods which provide useful categorization of some spam but are either too resource-intensive or have an unacceptably high false-negative rate for use in our production environment are good candidates for this kind of secondary screening. One simple option we will likely implement is to create and advertise a low-priority mail exchanger tarpit; these take advantage of the fact that a fair amount of spam software attempts to target low-priority mail exchangers with the assumption they are less heavily guarded than primary exchangers. Mail sent to this tarpit would be flagged for review, with the assumption that it is all spam. Any mails which Bogofilter was unsure about would be good candidates for training. Another option is to reimplement tools like Vipul's Razor on secondary hardware to scan a random sample of cached incoming mail. For legitimate mails, we will likely begin randomly sampling outgoing mail and flagging any messages which Bogofilter classifies as unsure or spam. Note that none of these tools would be added as active filtering mechanisms, but they could be useful outside of the mail flow in automating the process of keeping Bogofilter's wordlists up to date with the shifting nature of spam.

We will also continue to tweak the filter options as appropriate, and may make some of them dynamic. Possible options here include automatically modifying the Bogofilter configurations on the mail exchangers based on time of day to filter mail more aggressively during high spam periods such as weekends and late night, or at least to flag unsure messages during these times for scanning. Similarly, if the frequency of microspams increases and these become a problem we may lower `min_dev` to catch more of them on the first pass; another option is to make `min_dev` (or other parameters) dynamic based on message size, e.g., set `min_dev` to 0 for messages less than one kilobyte in size. It is entirely possible, however, that the filter will simply adapt; in this scenario things like normal local SMTP headers would become high enough spam indicators that microspams would always be blocked, while both short and long legitimate messages would get through due to their legitimate tokens. In any case, both of these types of dynamic configuration would probably only be necessary temporarily; eventually Bayesian mail classifiers will likely reach the point of using meta data – such as message size, arrival time, number and type of attachments, etc. – as spam/non-spam tokens.

In addition, we have done some investigation to see if we could use a lower base `spam_cutoff` and let even less spam through. Examining the classification of both the spams and legitimate mails that are being classified as "unsure," it looks like we would be safe dropping this value to 0.85 or even lower. However, we will need to do much more rigorous testing before we make such a fundamental change. We will also preferably have the above spot-checking of active mail and other measures in place to track any increase in the ongoing probability of false positives using this cutoff.

We would like to improve the caching system or, preferably, remove the need for it altogether. Storing emails for this kind of retrieval is not ideal. Recreating headers that Exchange has removed is inefficient at best and does not always work. At the least we would like to move to a system where each message can be flagged in a way that Exchange will not modify so that a simplified lookup will work, but this is not likely to be possible without adding something to the message body or requiring users to forward messages in a specific format. Both of these options are at odds with business standards. Other Bayesian classifiers such as DSPAM have experimented with keeping a record in the database of which tokens were present in a given message, but it is questionable whether this would scale to an environment such as ours.

Finally, we will need to reconsider rejecting spams during the SMTP exchange instead of silently dropping them. This gives spammers information about the delivery status of their messages which allows for the possibility of training an "evil" Bayesian filter on what messages our filter rejects or accepts. Given enough messages, this would allow spammers to craft messages specifically designed to pass through our filter. John Graham-Cumming has discussed this attack in detail [JGC]. However, this technique is easily defeated by simply not returning any information to spammers on the delivery status of their messages. This is non-ideal; it means that in that in the case of false positives the legitimate sender would not get an error to indicate their message was not received. However, the low false-positive rate demonstrated by this type of filtering likely justifies this inconvenience.

Related Work

At this time several other medium to large environments are known to be having success monitoring their mail flow with Bogofilter using single, centralized wordlists:

- York University in Toronto recently deployed Bogofilter as a classifier for their environment of 60,000 user accounts. Incoming mail volume is on the order of hundreds of thousands of messages per day. At the time of writing, this implementation was too new to have reliable numbers available, but early results are promising. In this implementation messages pass through Bogofilter

after DCC has already scanned them and rejected spams it can detect; approximately 30-40% of this remaining incoming mail is initially being flagged as spam by Bogofilter. This is an order of magnitude higher than the block rate of the SpamAssassin implementation that Bogofilter replaced, with a dramatically lower rate of false positives reported.

- A large ISP in Australia is using a modified version of Bogofilter with a single wordlist to watch 150,000 mailboxes. Over 1 million messages are processed per day. Bogofilter is believed to be around 95% effective in this environment, with no false positives reported in six months of operation. The wordlist management is completely centralized, with no user input whatsoever. Administrators keep Bogofilter's training current by manually scanning and training on random samplings of 100-300 "unsure" emails per week.

Both of these deployments have followed implementation and maintenance methodologies similar to the ones described in this paper.

Conclusion

Spammers have shown extreme willingness to adapt using any means at their disposal to spread their messages. Assumptions that they will be unwilling or unable to expend considerable resources or break the law to attack their targets are at best unfounded. Blocking techniques aimed at the mail routes and protocols are not directed at immutable properties of spam and therefore seem unlikely to succeed and are likely instead to drive spammers to more and more illicit methods and increasing collateral damage. Paul Graham was correct in noting that the content of spam is the one thing which cannot be changed arbitrarily; it can be altered and obfuscated, but at some point the content still must be there, or there is no potential for any return on the spammers' investment. Content-based filtering is therefore the best currently available approach to the problem, and our results show that Bayesian filters can be viable for the long term as adaptive content filters.

Our results demonstrate that, far from being just another failed attempt at producing a comprehensive and sustainable solution to the spam problem, Bayesian filters can be sufficient without aid from secondary methods. This is true even in large environments with central control, provided these filters are implemented carefully and with a solid understanding of the theory supporting them. There is a tremendous difference throughout the IT field between systems which truly can not stand the test of time and those which are simply difficult or tedious to implement correctly, and it is irresponsible to dismiss something with the latter property as though it had the former. We would all prefer it if we could block spam using easily implemented and foolproof methods, but if complicated solutions are required to win this fight, doing our homework seems the least we can do given the severity of the problem. Bayesian methods deserve further development, testing, and careful consideration before they are dismissed due to false assumptions or incomplete understandings of their value and effectiveness. Organizations and individuals are encouraged to implement similar solutions to the one detailed here and attempt to duplicate and verify our results.

Do we think Bayesian filters are a permanent solution to all spam? No. If nothing else, Moore's Law dictates that brute force attacks will eventually be viable. There is no reason to believe, however, that spammers will bother waiting that long just to implement high-cost attacks. Based on their previous patterns they are much more likely to attempt to sidestep the issue entirely and move to less direct methods: the spam equivalent of side-channel attacks. We are already seeing signs of these, as spammers are using browser malware to inject ads directly into web site content and are increasingly relying on worms to create zombie machines to send their spam for them. At the paranoid extreme, a fairly obvious convergence of these practices would be worms which inject ads directly into outgoing user emails, turning every legitimate mail into spam at the same time. The current malware epidemic demonstrates that such a methodology would be a much more cost-effective practice for spammers than escalating the filter war indefinitely. No filtering technology that only aims to block spam messages would be useful in stopping this type of spam, since any mail blocked would by definition be a false positive. SMTP replacements and authentication schemes would be similarly useless, since these messages would be sent through valid mail routes using appropriate credentials just as many worms are today. If and when we reach this point, however, we will no longer even be dealing with unsolicited commercial and bulk email. We will be dealing with something else. Filtering may be the most effective method for dealing with spam that is simply commercial email messages, but malware is something completely different.

Regardless of the possible future of spam, the most value offered by effective Bayesian filters is to be gained today while spammers are still unable to circumvent them and have not yet determined their next method of attack. The war against spam will no doubt continue to be fought with a range of methods across various platforms. No single technology will likely emerge which is capable of dealing with all attacks equally well. If the next phase does center around malware, it will only be won if we can get ahead of the spammers now and get a handle on the current virus epidemic. If Bayesian filters are at least robust enough to win the filtering battle for the foreseeable future they will be invaluable in buying administrators desperately needed time to move beyond filter maintenance to the much more serious problems of end-user

computer security. Instead of carelessly dismissing these filters as already beaten and wasting time pursuing weaker solutions, administrators should take advantage of the opportunities they provide to go on the offensive and finally gain an advantage over spammers before it is too late.

Availability

Vipul's Razor is available at http://razor.sourceforge.net/ .

Bogofilter is available at http://bogofilter.sourceforge.net/ .

qmail-qfilter is available at http://untroubled.org/qmail-qfilter/ .

Acknowledgements

We thank Paul Graham, Gary Robinson, and others for their pioneering work in this area. We especially thank Greg Louis and David Relson of the Bogofilter project for their responsiveness, feedback, and tuning suggestions throughout our initial training and implementation phases.

Author Information

Jeremy Blosser graduated from Indiana Wesleyan University in 1998 with a BS in political science, history, and philosophy. He moved to Texas to get a real job and has been a Web engineer and occasional system administrator for VHA, Inc. since 2000. He prefers playing with his three kids to sorting spam and can be reached at jblosser@vha.com or jblosser@firinn.org .

Raised by wolves, Dave Josephsen dedicated his life to feeding the naked and clothing the hungry. After a six-year stint in the United States Marine Corps, he became a tech support guru to the stars and eventually found his way to systems administration. He can be found in a server room near you, or contacted via email at djosephs@vha.com or dave@homer.cymry.org .

References

[ALL] Allman, Eric, *The State of Spam,* https://db.usenix.org/events/usenix04/audio/allman.mp3, June 2004.

[BAA] Baard, Mark, *Going Upstream to Fight Spam,* http://www.wired.com/news/print/0,1294,61971,00.html, January 2004.

[BOW] Bowers, Jeremy, *Spam Filtering's Last Stand*, http://www.jerf.org/iri/2002/11/18.html, November 2002.

[DOM] Domingos, Pedro and Michael Pazzani, *Beyond Independence: Conditions for the Optimality of the Simple Bayesian Classifier,* http://www.cs.washington.edu/homes/pedrod/papers/mlc96.pdf, 1996.

[EMM] Dreyfus, Emmanuel, *Mail-Filtering Techniques,* http://www.onlamp.com/pub/a/onlamp/2004/05/20/mail_filtering.html, May 2004.

[ESR] Raymond, Eric S., *2003 MIT Spam Conference: Lessons from Bogofilter*, http://www.usenix.org/publications/login/2003-06/openpdfs/spam.pdf, January, 2003. *In this talk Raymond explained his rationale for developing Bogofilter; he was primarily interested in putting a tool in the hands of end users since it was believed that Bayesian methods would not work in a centralized fashion. This point was summarized by Chris Devers in the June, 2003 issue of ;login magazine as follows: "As good as Graham's Bayesian algorithm is, ESR felt – as did many of the other speakers – that the nature of your spam/ham corpus is much more significant than the relative difference among any handful of reasonably good algorithms. (Back to the oft-repeated point about how corpus effectiveness falls apart when used for a group of users, as opposed to individuals.)"*

[GRA] Graham, Paul, *A Plan For Spam*, http://www.paulgraham.com/spam.html, August 2002.

[GRA2] Graham, Paul, *So Far, So Good,* http://www.paulgraham.com/sofar.html, August 2003.

[JAC] Jacob, Philip, *The Spam Problem: Moving Beyond RBLs,* http://theory.whirlycott.com/~phil/antispam/rbl-bad/rbl-bad.html, January 2003.

[JdeBP] Pollard, Jonathan de Boyne, *SPF is harmful. Adopt it.*, http://homepages.tesco.net/~J.deBoynePollard/FGA/smtp-spf-is-harmful.html, 2004.

[JdeBP2] Pollard, Jonathan de Boyne, *No anti-UBM measure for SMTP-based Internet mail works,* http://homepages.tesco.net/~J.deBoynePollard/FGA/smtp-anti-ubm-dont-work.html#Bayesian, 2004.

[JGC] Graham-Cumming, John, *2004 MIT Spam Conference: How to beat an adaptive spam filter,* http://www. jgc.org/SpamConference011604.pps, January 2004.

[JGC2] Graham-Cumming, John, *Fooling and poisoning adaptive spam filters,* http://www.sophos.com/sophos/docs/eng/papers/WP_PMFool_US.pdf, November 2003.

[KNO] Knowles, Brad, *Considered Harmful: SPF . . .*, http://bradknowles.typepad.com/considered_harmful/2004/05/spf.html, May 2004.

[LOU] Louis, Greg, *Tuning Bogofilter's Robinson-Fisher Method – a HOWTO*, http://www.bgl.nu/bogofilter/tuning.html, April 2003.

[LOU2] Louis, Greg, *Bogofilter Training: Comparing Full Training with Training-on-error,* http://www.bgl.nu/bogofilter/training.html, April 2003.

[LOU3] Louis, Greg, *Bogofilter Training: Comparing Full Training with Training-on-error, part 2* http://www.bgl.nu/bogofilter/training2.html, April 2003.

[LOU4] Louis, Greg, *Bogofilter Parameters: Effect of varying s and mindev, continued: Comparison of*

results from different email sources, http://www.bgl.nu/bogofilter/smindev3.html, April 2003.

[LOU5] Louis, Greg, *Is Bogofilter Scalable?* http://www.bgl.nu/bogofilter/scale.html, November 2002.

[LOU6] Louis, Greg, *Token Redundancy and the Effective Size Factor*, http://www.bgl.nu/bogofilter/esf.html, May 2004.

[MER] Mertz, David, *Spam Filtering Techniques: Comparing a Half-Dozen Approaches to Eliminating Unwanted Email*, http://gnosis.cx/publish/programming/filtering-spam.html, August 2002.

[PAG] Paganini, Marco, *ASK: Active Spam Killer* http://www.usenix.org/events/usenix03/tech/freenix03/full_papers/paganini/paganini_html/node2.html#SECTION00022000000000000000, April 2003.

[ROB] Robinson, Gary, *Spam Detection,* http://radio.weblogs.com/0101454/stories/2002/09/16/spamDetection.html, September 2002.

[ROB2] Robinson, Gary, *Handling Redundancy in Email Token Probabilities,* http://garyrob.blogs.com/handlingtokenredundancy94.pdf, May 2004.

[ROB3] Robinson, Gary, *Spam Filtering: Training to Exhaustion,* http://www.garyrobinson.net/2004/02/spam_filtering_.html, February 2004.

[SEL] Self, Karsten M., *Challenge-Response Anti-Spam Systems Considered Harmful*, http://linuxmafia.com/faq/Mail/challenge-response.html, December 2003.

[WAR] Ward, Mark, *How to Make Spam Unstoppable*, http://news.bbc.co.uk/1/hi/technology/3458457.stm, February 2004. *This article references John Graham-Cumming's work in defeating Bayesian filters using other Bayesian filters. While Graham-Cumming found this method can work, his conclusion was that it is very costly and quickly blocked. Nevertheless, many administrators and weblogs continue to point to this experiment (and specifically this article) as "proof" that Bayesian filters can be easily defeated.*

[ZDZ] Zdziarski, Jonathan A., *Bayesian Noise Reduction: Progressive Noise Logic for Statistical Language Analysis,* http://www.nuclearelephant.com/projects/dspam/bnr.html, February 2004.

Appendix A

The scripts below are what we used to create our initial wordlists and tune Bogofilter. This functionality can now be found in the scripts that ship with Bogofilter, but these are provided for illustration purposes and in the hope that they may be useful.

```
#!/bin/sh

# retrain.sh: Test various bogofilter.cf values. Create randomized message
# lists, fully train on 10k each spam and non-spam, train on error for other
# messages, test on 5k each spam and non-spam. Repeat until all messages are
# trained on error twice, then output stats for each test. Log everything.

# syntax: retrain.sh <logfiletag>

cd /usr/share/bogofilter/retrain/

echo "removing old wordlists..."
rm -f data/{good,spam}list.db

if [ ! -f list ]; then
        echo "making lists..."
        ./makelists.sh 1>&2
fi

echo "doing 10k spam..."
while read
do
        echo "${REPLY}" 1>&2
        < "${REPLY}" ./bogofilter -C -c ./bogofilter.cf -s
done < 10k_spam_list.random

echo "doing 10k notspam..."
while read
do
        echo "${REPLY}" 1>&2
        < "${REPLY}" ./bogofilter -C -c ./bogofilter.cf -n
done < 10k_notspam_list.random

echo "doing rt round 1..."
./rt.sh rest_list.random | tee ./rt."${1}"-1.log | sed -f ./rt.sed 1>&2

echo "doing tt round 1..."
./tt.sh 5k_spam_list.random | tee ./tt."${1}"-1.log | sed -f ./rt.sed 1>&2
```

```
./tt.sh 5k_notspam_list.random | tee -a ./tt."${1}"-1.log | sed -f ./rt.sed 1>&2

echo "doing rt round 2..."
./rt.sh rest_list.random | tee ./rt."${1}"-2.log | sed -f ./rt.sed 1>&2

echo "doing tt round 2..."
./tt.sh 5k_spam_list.random | tee ./tt."${1}"-2.log | sed -f ./rt.sed 1>&2
./tt.sh 5k_notspam_list.random | tee -a ./tt."${1}"-2.log | sed -f ./rt.sed 1>&2

echo "adding tt messages round 1..."
./rt.sh 5k_list.random | tee ./rt."${1}"-3.log | sed -f ./rt.sed 1>&2

echo "doing tt round 3..."
./tt.sh 5k_spam_list.random | tee ./tt."${1}"-3.log | sed -f ./rt.sed 1>&2
./tt.sh 5k_notspam_list.random | tee -a ./tt."${1}"-3.log | sed -f ./rt.sed 1>&2

echo "adding tt messages round 2..."
./rt.sh 5k_list.random | tee ./rt."${1}"-4.log | sed -f ./rt.sed 1>&2

echo "doing tt round 4..."
./tt.sh 5k_spam_list.random | tee ./tt."${1}"-4.log | sed -f ./rt.sed 1>&2
./tt.sh 5k_notspam_list.random | tee -a ./tt."${1}"-4.log | sed -f ./rt.sed 1>&2

echo "doing stats..."
echo "1st run:"
./getstats.sh ./tt."${1}"-1.log
echo "2nd run:"
./getstats.sh ./tt."${1}"-2.log
echo "3rd run:"
./getstats.sh ./tt."${1}"-3.log
echo "4th run:"
./getstats.sh ./tt."${1}"-4.log
```

==========

```
#!/bin/sh

# makelists.sh: Generate randomized message lists needed by retrain.sh.

# This script requires rl, found here:
# http://tiefighter.et.tudelft.nl/~arthur/rl/

find /var/spam/corpii/{NOTSPAM,SPAM} -type f > list
< list rl > list.random
< list.random grep -i '/corpii/spam' | head -10000 > 10k_spam_list.random
< list.random grep -i '/corpii/notspam' | head -10000 > 10k_notspam_list.random
cat 10k_spam_list.random 10k_notspam_list.random | rl > 10k_list.random
< list.random grep -i '/corpii/spam' | tail -5000 > 5k_spam_list.random
< list.random grep -i '/corpii/notspam' | tail -5000 > 5k_notspam_list.random
cat 5k_spam_list.random 5k_notspam_list.random | rl > 5k_list.random
sort 10k_list.random 5k_list.random | grep -v -F -f - list.random | \
                                                  rl > rest_list.random
```

==========

```
#!/bin/zsh

# rt.sh/tt.sh: Process messages specified in provided file through Bogofilter.
# As tt.sh, output Bogofilter's classification.  As rt.sh, output
# classification and train Bogofilter on error.

BOGOFILTER="/usr/share/bogofilter/retrain/bogofilter"
BOGOFILTERCF="/usr/share/bogofilter/retrain/bogofilter.cf"

< "${1}" | while read
do
    printf "%41s: " `echo "${REPLY}" | sed -e 's%/var/spam/corpii/%%'`
    REAL=`echo "${REPLY}" | sed -e 's%/var/spam/corpii/\([^/]\+\)/.*%\1%'`
    printf "%7s: " "${REAL}"
    BOGO=`< "${REPLY}" "${BOGOFILTER}" -C -c "${BOGOFILTERCF}" -v`
    GUESS=`echo "${BOGO}" | sed -e 's/.*\(Spam\|Legit\|Unsure\).*/\1/'`
    SPAMICITY=`echo "${BOGO}" | sed -e 's/.*\(spamicity=[^ ]\+\),.*/\1/'`
    printf "%6s: " "${GUESS}"
    printf "%20s\n" "${SPAMICITY}"
    if [[ "`basename ${0}`" == "rt" ]]; then
        case "${GUESS}" in
            "Spam")
                if [[ "${REAL}" == "NOTSPAM" ]]; then
                    < "${REPLY}" "${BOGOFILTER}" -C -c "${BOGOFILTERCF}" -n
                fi
            ;;
            "Legit")
                if [[ "${REAL}" == "SPAM" ]]; then
                   < "${REPLY}" "${BOGOFILTER}" -C -c "${BOGOFILTERCF}" -s
                fi
            ;;
            "Unsure")
                if [[ "${REAL}" == "NOTSPAM" ]]; then
                    < "${REPLY}" "${BOGOFILTER}" -C -c "${BOGOFILTERCF}" -n
                else
                    < "${REPLY}" "${BOGOFILTER}" -C -c "${BOGOFILTERCF}" -s
                fi
            ;;
        esac
    fi
done
```

```
# rt.sed: Colorize output of rt.sh.

s/\( SPAM\):/^[[1;31m\1^[[0m:/
s/\(NOTSPAM\):/^[[32m\1^[[0m:/
s/\(Spam\)/^[[1;31m\1^[[0m/g
s/\(Legit\)/^[[32m\1^[[0m/g
s/\(Unsure\)/^[[7;34m\1^[[0m/g
```

```
#!/bin/sh

# getstats.sh: Output summary of results from retrain.sh-generated logs.

echo "false positives: `grep 'NOTSPAM:   Spam:' ${1} | wc -l`"
echo "false negatives: `grep ' SPAM:  Legit:' ${1} | wc -l`"
echo "unsure spams   : `grep ' SPAM: Unsure:' ${1} | wc -l`"
echo "unsure notspams: `grep 'NOTSPAM: Unsure:' ${1} | wc -l`"
```

Appendix B

The scripts below are used for filtering our messages during the SMTP exchange.

```
#!/bin/sh

# qq-qfilter: log the mail, then add and fix* the Bogofilter header, then
# forward the mail to the caching server, then check if it's spam and refuse it
# if it is.

# A full description of qmail-qfilter's operation is beyond the scope of this
# paper, but in summary, it takes a mail message on stdin and pipes it through
# a "--" delimited list of filters.  Each filter's stdout becomes stdin for the
# next filter.  Exit codes are checked and manipulated to provide the calling
# qmail daemon what it expects.

# *"Fixing" the Bogofilter header means making sure it is the first line in the
# header.  The version of Bogofilter we use does not place this header
# consistently; this has reportedly been fixed in current versions.

exec /usr/bin/qmail-qfilter /usr/local/bin/qfilter-logger -- \
      /usr/bin/bogofilter -l -e -p -- \
      /bin/sed -e '1,/^X-Bogosity:/{;' -e '/^X-Bogosity:/!{; H; d; };' \
                  '/^X-Bogosity:/{; p; g; D; }; }' -- \
      /usr/local/bin/qfilter-cache -- /usr/local/bin/qfilter-spamcheck
```

==========

```
#!/bin/sh

# qfilter-spamcheck: read the first line of stdin as an X-Bogosity header; if
# the message is spam, exit 31 to refuse delivery, otherwise pass the message
# through unaltered.

# This script requires rewind, which of part of DJB's serialmail package and
# can be found here:
# http://cr.yp.to/serialmail.html

if head -n 1 | grep -q '^X-Bogosity: Spam,'; then
   exit 31
fi

/usr/local/bin/rewind

cat -
```