MARC WALLMAN

spam filtering for the enterprise

Marc Wallman is Senior System Administrator at North Dakota State University.

Marc.Wallman@ndsu.edu

THE EVER INCREASING VOLUME OF spam causes ubiquitous frustration for end users. In the spring of 2005, North Dakota State University (NDSU) made a commitment to do something about this problem for its users. What follows is a presentation of our response: a spam filtering system combining existing open source software and homegrown applications. The resulting system was designed to have a minimal impact on our existing mail system, to be easily scalable, and to be modular enough for us to be able to perform maintenance on particular components without disrupting the others.

The spam filtering system presented here is not a stand-alone system. It was assembled from existing open source products and integrated using homegrown connectors into an existing enterprise email system. Because this is an integrated solution, the spam filter and the mail portion of the system will both be presented here. The email system, and to a certain extent the spam component, developed organically along the lines of the spiral model of software engineering (although much more informally). The focus of this article will be on the design of the system as a whole and the implementation of the spam filter.

Why create a spam filtering system when there are hundreds of open source products that address the problem of spam? Why not just pick one and use it? Here's why: there is no single open source product that is appropriate in and of itself at the enterprise level. Existing open source spam products are partial rather than comprehensive in their approach. They provide specific solutions to particular problems. For instance, SpamAssassin [1] is a very effective tool, but making it directly available to end users is not appropriate. System administrators may be comfortable with Bayesian filtering, editing rule sets, and configuring delivery rules, but end users are not. SpamAssassin is useful, but incomplete without help from other applications.

An important caveat exists here. Deploying solutions like SpamAssassin may be possible in a straightforward way if end users do not need to be given a choice of whether or not they use it or how it behaves. Consider the case of EduTech, the state-funded IT organization that provides email services for K–12 institutions in the state of North Dakota. Clearly, much of the spam that floats around the

Internet is inappropriate for minors to receive; there is no need to give them an option about how the spam they receive is handled. EduTech system administrators may decide what is spam and what is not spam. Anything that is determined to be spam, they may delete. No flexibility is required on the part of the EduTech system administrators.

NDSU hosts approximately 18,000 IMAP mail accounts for six schools in the North Dakota University System plus the University System Office. It accepts approximately 135,000 messages on a given weekday for approximately 210,000 recipients. A typical message takes 18 seconds or less from the time it is sent to the time it is read by an actively checking recipient. A middleware solution called Hurderos was developed in-house for integrating mail for the aforementioned institutions. Unfortunately, this middleware system is beyond the scope of this article. Those who are interested may read more about the GPL-licensed Hurderos at http://www.hurderos.org. For the sake of simplicity, the rest of this article will treat this spam solution as a single-institution solution.

A Failed Attempt and Lessons Learned

This section will briefly describe a first attempt at a SpamAssassin-based spam solution and some of the lessons learned. An attempt had been made to provide a central solution several years prior. However, the implementation of a centralized SpamAssassin-based solution was largely ineffective, because insufficient thought was given to maintainability and scalability. This opt-in service worked by piping mail through SpamAssassin before final delivery on the IMAP mail servers where users' mailboxes resided. Two main problems existed with this system. First, SpamAssassin was invoked once per piece of mail, resulting in lots of expensive forking. Batch processing was not possible. Processing the bounces from a bulk mailing to the entire campus would cause noticeable slowness on the server that housed the sender's mail account. Second, as SpamAssassin aged, it became difficult to update. The solution was implemented using a 2.x release of SpamAssassin, which required Perl 5.6. A little over a year later, SpamAssassin 3.0 was released, which required Perl 5.8. The IMAP mail servers ran RedHat Enterprise Linux 2.1, which provided only Perl 5.6. Suddenly, we were in a situation where we would have to maintain our own version of Perl and associated modules if we wanted to keep SpamAssassin up to date. SpamAssassin requires frequent updates and architectural changes to the system it runs on, while the University of Washington IMAP daemon has the opposite requirements.

As the SpamAssassin-based solution aged and lost its effectiveness, some users turned to the built-in spam filtering capabilities of their email clients. While possibly effective on a case-by-case basis, this approach was never successful at an enterprise level, for two main reasons. First, we did not have, and still do not have, established mail client standards; therefore, our help desk ends up supporting everything from Outlook to Eudora to Pegasus Mail. This is an unfortunate situation, but one that many other higher-education institutions and ISPs find themselves in. Client diversity presented a serious obstacle to providing effective end-user support for client-side spam filtering. Further, client-side spam filtering is not universally present in these various mail clients and is complicated by users who use multiple mail clients (e.g., Thunderbird in the office and a Webmail client on the road). Second, this strategy places the work of catching spam on the end user. For at least some users, this is a difficult thing to do. It is not uncommon for people to misconfigure their mail client to block messages from everyone on campus.

System Requirements

The requirements for this system came from a committee of users led by NDSU's IT Security Officer (ITSO). This committee established general requirements for this new spam filtering system. This non-technical component was critical to the success of this project. The recommendations were just that—recommendations. No particular solution was specified (commercial or open source). The recommendations were to be fulfilled insofar as was possible. The rest of this section summarizes these recommendations.

Mail should be grouped into three categories: (1) obvious spam, (2) potential spam, and (3) not spam. Identifying what is spam is not an exact science. The intent of these three categories is to recognize this fact. Messages falling into the category of obvious spam are those that are most clearly identifiable as spam and most likely to be considered spam by everyone. Potential spam consists of those messages that are probably spam. It is a more aggressive category than obvious spam and thus encompasses a larger corpus of messages. The not spam category is self-evident.

Using these categories, different levels of service were established. At all levels of service, mail that is considered not to be spam is delivered to users' inboxes. Obvious and potential spam are handled differently, depending on the desired level of service. A matrix summarizing the various levels of service is depicted in Table 1.

Level	*Obvious Spam*	*Potential Spam*
Disabled	Deliver	Deliver
1	Quarantine	Deliver
2	Quarantine	Quarantine
3	Delete	Quarantine
4	Delete	Delete

TABLE 1: RECOMMENDATIONS FOR HANDLING SPAM

By default, new users have the spam service disabled, making this an opt-in solution. All mail is delivered to users' inboxes, regardless of the category it falls into. At level 1, obvious spam would be quarantined and potential spam would be delivered to the user's inbox. Ideally, the quarantine would appear as another folder under the user's mail account. Both potential spam and obvious spam are quarantined at level 2. At level 3, obvious spam is simply deleted with potential spam being quarantined. Finally, at level 4, all spam is deleted. In addition to these varying levels of service, the committee asked that users be able to specify lists of senders who were considered safe. Mail from addresses on this safe sender list was always to be delivered. A corresponding list of senders to block was also requested. Mail from addresses on this list was always to be deleted.

Open Source Spam Filtering

The open source solution deployed at NDSU was based on two products already in use, MailScanner and SpamAssassin. This section will present the resulting solution.

Spam filtering for the enterprise was developed within the context of a preexisting open source email infrastructure. The mail system is an IMAP mail solution built on Linux. Each of the components—sending, delivery, and retrieval—will be briefly examined. It is illustrated below in Figure 1.

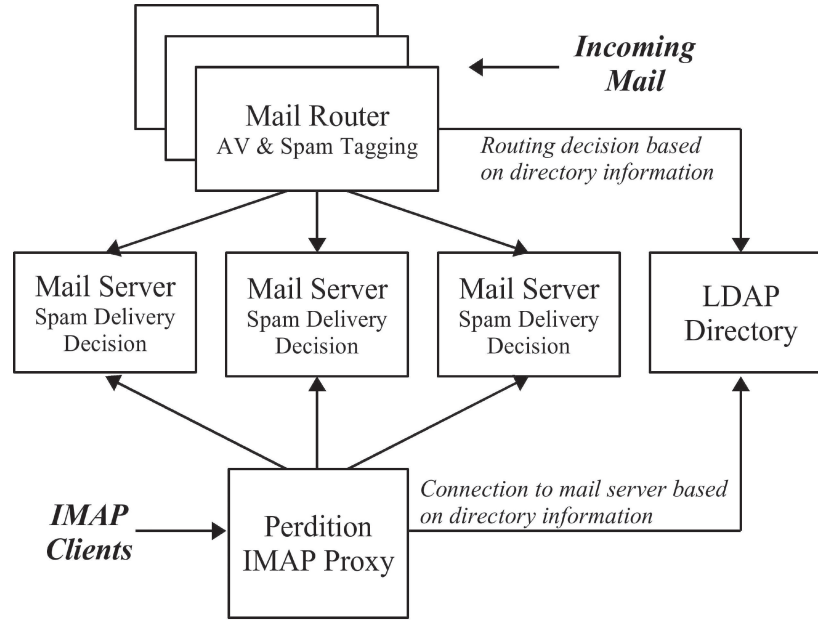

FIGURE 1: EXISTING MAIL INFRASTRUCTURE

All outgoing mail is handled by a single SMTP server running Sendmail on Linux. This portion of the mail system was unchanged throughout this entire process. To date, no spam, or even virus, filtering takes place on this server. All such filtering occurs only on mail delivery. (This is all that is necessary to protect our own users from spam and viruses.) It is likely that this will change in the future. Spam originating from our institution and other institutions within the North Dakota University System has caused no end of headaches for campus security officers who handle spam complaints and oversee the cleanup of compromised desktops, which are almost always the source of these problems.

The process of mail delivery begins with the mail routing servers that run Sendmail and MailScanner [2]. Mail routers are depicted in Figure 1 in a stacked configuration because, conceptually, there need only be one, but multiple configurations may be used to distribute the processing load of the incoming mail. At NDSU this is done by using multiple MX entries in our DNS [3]. One could also accomplish this by using a load balancer, such as the Linux Virtual Server [4].

Routing information used by the routing servers is stored in an LDAP directory using the standard Sendmail schema. MailScanner controls virus scanning and spam tagging. The delivery process works as follows:

A Sendmail daemon running on the incoming mail servers receives a connection from an MTA requesting that mail be delivered to a particular user or users. Sendmail looks up the address(es) in the LDAP directory to determine where the mail should be routed. Routing rules appear in the directory like this:

```
mailLocalAddress: Marc.Wallman@ndsu.edu
mailRoutingAddress: mwallman@imap3.ndsu.edu
```

Making a routing decision at the point of entry is very important. Spammers routinely use brute force as a method of delivering spam. They will crawl through a name dictionary constructing possible usernames with the hope of finding one that will be accepted by the targeted domain. If the incoming mail servers are relay only, they do not know what usernames are valid for hosts or domain they serve and must blindly accept all messages for these hosts and domains. Much of the spam coming from brute force ends up stuck in the queues. The destination hosts reject all the messages for unknown users and the spammers won't take the bounces back. Queues become bloated with accumulated bounces. When the server gives up on delivery for these bounces, new

ones quickly take their place in the queues. Initially we had our incoming mail server set to relay and not route. It was not uncommon to have a total of 15,000 bounces stuck in our queues.

Messages that Sendmail accepts are delivered to a special queue directory that is processed by MailScanner, which in turn controls virus scanning and spam flagging. MailScanner is highly configurable and scalable. It uses one or more external virus scanners (there are many to choose from—we use McAfee [5]) to catch viruses and SpamAssassin to filter for spam). MailScanner is highly configurable and very efficient. Virus scanning and spam filtering happen in batch mode, not on a per-message basis. The system administrator may determine the batch size and scan interval.

MailScanner may make a decision on how to handle mail based on a message's spam score as rated by SpamAssassin. The score is assigned based on the application of many weighted spam tests. The point value of all tests that fail (e.g., a forged sending IP address) are added up and the message is given a total score. We chose to only tag mail with X headers. A decision about what to do with the mail is not made at this level, because it cannot be made on a per-user basis.

Messages were tagged to match the recommendations of our user community. Obvious and potential spam were each given their own tags:

```
X-NDUS-SpamFlag: Obvious Spam
X-NDUS-SpamFlag: Potential Spam
```

Mail given 8.0 or more points by SpamAssassin was tagged as obvious spam. Mail assigned between 5.0 and 8.0 points was tagged as potential spam. Mail scoring less than 5.0 was not tagged. The raw score is also embedded in the header. Category tags are added at this level for later use in deciding how to deliver mail based on an individual user's chosen level of service. Using category tags instead of a raw SpamAssassin score allows flexibility in how aggressive we wish to be in categorizing mail. (A small contingent of GroupWise users exist at NDSU. A special tag, `X-SpamFlag: Yes`, was also added to all messages that were considered spam, i.e., both obvious and potential spam. GroupWise is able to make use of this special tag with its client-side junk mail processing.)

Once MailScanner completes scanning and tagging, it re-queues the mail for final delivery by Sendmail. The final delivery point is the routing address specified in the mailRoutingAddress attribute LDAP directory. We happen to allow users to opt out of our mail solution altogether and have their mail routed to any address they choose. Common destinations are hotmail.com, yahoo.com, and gmail.com. Mail bound for these and other off-campus destinations is still tagged. If they choose, users may do client-side filtering based on the headers we add.

After going through the incoming mail router, messages are passed to their final delivery point: a server running the University of Washington IMAP daemon [6]. Here, procmail is invoked to look for the spam tags and make a delivery decision. Four different templates exist, corresponding to the four levels of service outlined above. When users select a level of service, the appropriate template is installed in the user's `.procmailrc` file in their home directory. Based on the recipe in `~/.procmailrc` and the categorization of the message (obvious spam, potential spam, or not spam), incoming mail is either delivered, quarantined, or deleted (i.e., delivered to `/dev/null`). The quarantine is simply an IMAP folder that exists in the user's email account. We have chosen to name this folder SPAM-Quarantine. Safe-sender and block-sender lists as described above are also encoded in procmail rule sets. The following is an example of a procmail recipe for level 2 (quarantine/quarantine):

```
INCLUDERC=.SafeSenderList
INCLUDERC=.BlockSenderList
```

```
:0 W
* ^X-NDUS-SpamFlag: Potential Spam
|/usr/local/sbin/dmail +SPAM-Quarantine
:0 W
* ^X-NDUS-SpamFlag: Obvious Spam
|/usr/local/sbin/dmail +SPAM-Quarantine
```

As indicated in this recipe, the safe-sender and block-sender lists are stored in separate recipe files. The following is an example of an entry from a block sender list:

```
:0:
* ^From:.*foo@example.com
/dev/null
```

Readers may learn more about procmail at http://www.procmail.org. The dmail mail delivery agent referenced above in the procmail recipes is bundled with the University of Washington IMAP daemon. It is used instead of the native delivery agent in procmail in order to allow us to use MBX-style indexed mailboxes. Documentation on dmail is included with the UW-IMAPd.

Mail retrieval is mediated by the Perdition Mail Retrieval Proxy [7]. Perdition mediates all communication with the back-end IMAP servers (it also supports POP, which we do not use). When a user initiates an IMAP login, Perdition takes the supplied username and looks in LDAP to determine the server on which the user's mailbox resides. All subsequent communication from the client is simply proxied over to the real mail server. Similarly, responses from the mail server are proxied back to the client. In this way, many mail servers may be used to deliver IMAP service, while the end user is blissfully unaware of this as they only connect to the proxy. Perdition is not a resource-intense application. A single 2-CPU Linux server with 1GB of RAM easily proxies all IMAP communication for our system. Typical weekday traffic is over 110,000 logins from just under 9,000 unique users. If there were ever a need, this portion of the system could easily be scaled by adding servers running Perdition in a load-balanced configuration.

The remaining portion of this system is the mechanism by which users enable/disable their spam filter and populate their safe- and block-sender lists. This was integrated with a preexisting Web application that allows users to enroll for new services and do some limited management of services they are subscribed to (e.g., users wishing to have their @ndsu.edu email forwarded to their Hotmail account would use this site to do so). The design of the user interface was taken almost verbatim from the recommendations of the user community. The presentation of the form that allows users to enable the various levels of spam filtering service described above is basically the same as what is shown in Table 1. Simple Web-based lists allow the population, or depopulation, of the safe and block lists. Submission of the spam filtering service level form or safe/block sender lists triggers a program to execute on the mail server that hosts the submitter's mail account: the appropriate procmail template is placed in the user's home directory, the SPAM-Quarantine folder is created if it does not exist and is added to the list of visible folders if it is not already there, and the safe/block lists are recreated with the appropriate values. Since all of these items represent files in the user's home directory, they are all susceptible to being deleted. The decision to have these Web forms recreate rather than update the files was made so that repair of a broken spam filter was something the end user could do. (The spam filter most commonly gets broken when users delete their SPAM-Quarantine folder.) This strategy results in many fewer support tickets being sent from help desk staff to system administrators.

Measures of Success

From the perspective of the system administrator, the successful implementation of this spam solution arises from two key sysadmin-friendly traits: maintainability and scalability.

The system's maintainability derives from a number of factors. First, it is modular. The first attempt at implementing a spam filtering solution co-located the mail accounts with the spam filter's intelligence. This created a functional dependency that impaired our ability to perform upgrades. SpamAssassin required frequent upgrades, whereas the IMAP daemon required few. Now these applications reside on separate systems and the use of the "potential" and "obvious" tags further the insulation of these applications by establishing a kind of API. The end result of the spam filtering system is only to assign category tags to pieces of email. The mail servers process spam simply by examining the tags contained within the email messages.

Second, the system is simple. The modularity not only removed functional dependency, it also removed complexity. Upgrades to the mail server may now be done largely without consideration for how spam is identified. Upgrades to the spam solution may be done largely without consideration for how the mail servers are configured. Another respect in which this system exhibits simplicity is in its functionality. Only four levels of service exist (five if you count the disabled state). All are clearly defined and sufficiently abstracted from the process of categorization to allow a great deal of flexibility with regard to how this categorization is accomplished. This flexibility should also be of benefit as new techniques become available to combat spam. There is a reasonable chance that the MailScanner/SpamAssassin solution could be swapped out with another technology at some point in the future if the need arose.

Third, upgrades to the incoming mail servers which tag spam do not require downtime. As previously mentioned, multiple mail routers may exist and they are redundant. More may be added or existing systems may be removed. As long as sufficient throughput exists to run in a degraded mode—that is, with at least one routing server offline—it is possible to take systems out of production for upgrade and place them back when the upgrade is complete. Being able to do maintenance on production systems is a very good thing for system administrators. There are enough other opportunities for work in the early morning hours and on weekends.

In addition to maintainability, this system is also scalable. If the load processing for incoming spam becomes too great for existing routing servers, just add more. A side benefit is that more servers means not only increased throughput but more redundancy. At NDSU, we currently have three mail routers in production. We almost never see high load on all three servers simultaneously. In those rare instances that we do get alerts about load across all systems, they have invariably recovered by the time the situation is investigated. All other parts of this system are scalable too. Perdition mail proxies and outgoing mail servers may be added in a load-balanced configuration. IMAP mail servers may also be added, with the caveat that user accounts must be migrated; methods exist, however, for doing this with minimal disruption to the running systems. We have developed code that automatically coordinates the transfer of users' mail accounts between back-end mail servers with the population of updated mail routing information in our LDAP directory.

With all this said, how well do we do at catching spam? How well can we do? Figure 2 shows the amount of spam identified per week as a percentage of our total email.

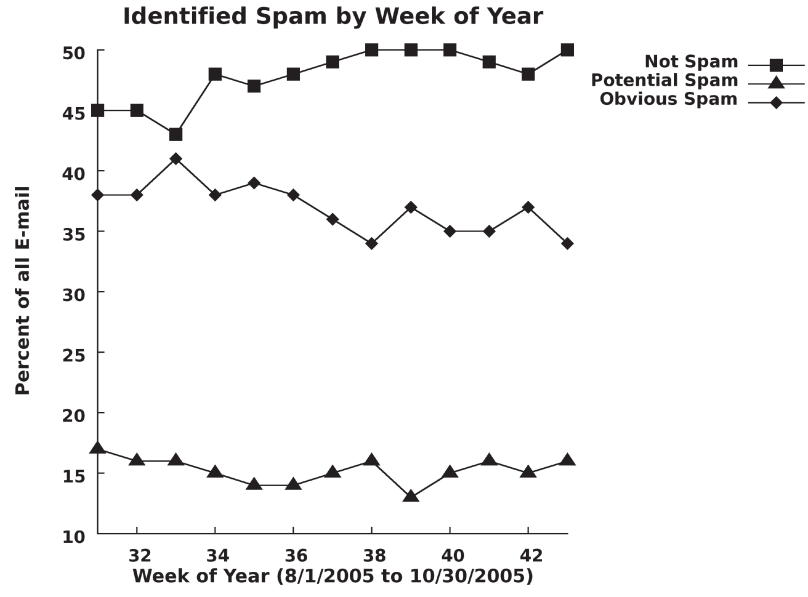

FIGURE 2: IDENTIFIED SPAM BY WEEK

The data depicted here is for the week beginning August 1, 2005 (week 31), through the week beginning October 24, 2005 (week 43). The graph has been aggregated by week in order to make it more readable. Legitimate mail decreases over the weekend, while spam seems to remain constant. As the graph shows, we routinely identify just over 50% of our incoming mail as spam. If measured against some reports, the spam filter does not appear to be very effective. Symantec recently reported that 61% of all email is spam [8]. Others have reported the percentage of spam to be much higher. User feedback has been quite positive (see below); so what might account for this apparent discrepancy in the numbers? We may simply be below average in the amount of spam that we receive, or the positive user response may be mainly due to the relative improvement offered by the new system. They may be happy that they are getting less spam, but we could be doing better. Third, the data sample shown in Figure 2 shows a slight decrease in the effectiveness of our spam filter since the beginning of August. SpamAssassin 3.1.0 was released on September 14, 2005, and, as of this writing, has not been put into production here at NDSU. SpamAssassin's effectiveness decreases with age. Spammers actively work to find ways to get their messages through without being identified. This implies that the numbers for September 2005 represent a low point of effectiveness. It is likely that the effectiveness of this solution will continue to decrease until SpamAssassin is upgraded. Last, we have not been overly aggressive in our attempts to tag spam. We have not used SpamAssassin's rules du jour, attempted to develop our own rule sets, or adjusted the thresholds at which we tag potential and obvious spam. It is likely we could do far better, but our users have been satisfied, so we have focused on other things.

During the summer of 2005 we had about 50 testers who used their accounts to quarantine all mail tagged as obvious or potential spam. Feedback from this group was VERY positive. All were extremely impressed at how well the system functioned. Testers also reported that there were almost no pieces of legitimate mail that were being tagged as spam, while a small amount of difficult-to-catch spam did get through. We could be more aggressive in our filtering.

The system was officially launched on September 19, 2005, supported by a considerable amount of PR. Oddly, far fewer people than expected signed up. Subscription rates jumped to about 7% after the initial launch, with a very minor

growth trend. Feedback from the user community continues to be good. There has been quite a bit of unsolicited praise, which is very unusual. Unsolicited complaints or silence is more in line with the norm. Does most of our spam go to a small portion of the user community, leaving the majority unaffected? Are more people using client-side tools for addressing spam than we suspect? At this point, we do not know.

Future Directions

Two main questions confront us with this system. The first has already been mentioned: Why are there so few subscribers? The second is: Can we keep up with the spammers? Spam filtering differs greatly from Web or calendar service, for example, in that people are actively attempting to make spam filters obsolete. I believe this system has been built with sufficient flexibility so that it will be able to adapt, but that remains to be seen.

REFERENCES

[1] SpamAssassin, from the Apache Software Foundation: http://spamassassin.apache.org/.

[2] http://www.sng.ecs.soton.ac.uk/mailscanner/.

[3] For more information on how MX records work, see section 5 of RFC 2821 (Simple Mail Transfer Protocol).

[4] http://www.linuxvirtualserver.org/.

[5] The North Dakota University System has a licensing agreement with McAfee, which made this antivirus scanner an easy choice. ClamAV represents an open source alternative to commercial antivirus software: http://www.clamav.net/.

[6] http://www.washington.edu/imap/.

[7] http://www.vergenet.net/linux/perdition/.

[8] "Symantec Internet Security Threat Report Identifies Shift Toward Focused Attacks on Desktops," September 16, 2005: http://www.symantec.com/press/2005/n050919a.html.