USENIX Association

Proceedings of LISA 2002: 16th Systems Administration Conference

Philadelphia, Pennsylvania, USA
November 3–8, 2002

Pp. 13-20 of the *Proceedings of LISA '02: Sixteenth Systems Administration Conference* (Berkeley, CA: USENIX Association, 2002).

Spam Blocking with a Dynamically Updated Firewall Ruleset

Deeann M. M. Mikula, Chris Tracy, and Mike Holling – Telerama Public Access Internet

ABSTRACT

In this paper, we detail our methods for controlling spam at a small ISP, reducing both resource usage and customer complaints. We will discuss our initial unsuccessful tactics, and the resulting development of our unique spam blocking system. Deny-Spammers classifies hosts as probable spammers and inserts those hosts into a dynamically updated firewall ruleset on our mail server, thereby effectively blocking the host from making an SMTP connection to our mail server. Our analysis demonstrates that this has been effective in reducing the amount of spam that our customers receive, and the burden on our limited resources.

Introduction

Are you aggravated by spammers launching what are effectively Denial of Service Attacks against your mail server? We were, and after several attempts at using some established spam-control techniques, we recognized the need to create our own novel approach, which we affectionally call "Deny-Spammers."

With the abundance of spam on the Internet today, nearly every ISP finds themselves forced to participate in some kind of spam[1] blocking.

Jon Postel wrote in RFC 706 [1], "there is no mechanism for the [email] Host to selectively refuse messages. This means that a Host which desires to receive some particular messages must read all messages addressed to it. Such a Host could be sent many messages by a malfunctioning Host. This would constitute a denial of service to the normal users of this Host. Both the local users and the network communication could suffer."

While this scenario is common enough today, it was a shocking thought in 1975 when Postel authored RFC 706. Then, the Internet was still a place of cooperation where users operated with the "greater good of the net" in mind. Today's mail servers operate under an almost constant threat of Spam Denial of Service attacks.

In an article in The New York Times from June 27, 2002, Jennifer Lee writes, "Brightmail, which maintains a network of In boxes to attract spam, now records 140,000 spam attacks a day, each potentially involving thousands of messages, if not millions." [2] A similarly bleak report from Hotmail states that 80% of its almost two billion processed email messages are spam [3].

Of particular interest to us as an ISP is the reaction of the customer base to spam in their inbox. A report by Gartner Consulting states that 53% of its respondents place the blame for spam on their ISP. They found that UCE (Unsolicited Commercial Email) ranks fourth in reasons for customer churn [4].

For these reasons, having a spam control policy is no longer an option for an ISP, no matter what its size. Hotmail subscribes to MAPS (Mail Abuse Prevention System) RBL (Realtime Blackhole List) [5]. Both AOL and Earthlink advertise their spam filtering as a benefit to their services.

Telerama is a small ISP, established in Pittsburgh, PA in 1991. Our mail system consists of a single server for both incoming and outgoing mail. It uses a 1 GHz Athlon processor and has 640 MB of RAM, running FreeBSD 2.2.8-STABLE. Our mail transport agent is *qmail-1.03* [6]. In addition to the stock *qmail* distribution, we are using the *qmail-uce checklocal* patch [7] to reject mail for non-existent mailboxes.

This server handles all incoming and outgoing mail for approximately 7,000 accounts, including well over 600 hosted virtual domains, and their associated addresses. The server typically delivers between 50,000 and 70,000 incoming e-mail messages to local users in a 24-hour period. Attempted deliveries, including messages to non-existent mailboxes, varies between 100,000 to 140,000 messages in the same time period. We approximate that 50% of the attempted deliveries are blocked by the *qmail-uce* checklocal patch alone.

From the user's perspective, spammers cause a general slowing down of our entire mail system. At times, a single spammer would open hundreds of simultaneous SMTP connections to our mail server, dumping thousands of messages into our mail queue. This causes delays in message delivery lasting from minutes to hours. A user sending mail through our system could wait up to 30 seconds before a 220 response [8] code is returned by the SMTP server. As most spam is generated during business hours, the mail queue would

[1] "Spam" is the term commonly used to refer to mass-emailed, unsolicited commercial email (also known as UCE), sent by a person or organization usually referred to as a "spammer."

shrink back to manageable sizes in the evening. The next day, the problem repeats itself. When the snappy performance we are used to diminishes, our users start to complain about the sluggish performance. We wanted to be able to identify spammers and simultaneously block them in order to prevent degradation of the performance of the mail server.

What We Tried First

Two alternate approaches we attempted before developing Deny-Spammers were:

- using *qmail-uce*'s checklocal patch to deny mail for non-existent mailboxes
- using ucspi-tcp's *rblsmtpd* [9] in conjunction with several RBL sources

First, we attempted to implement the *qmail-uce* checklocal patch alone. This patch makes *qmail* reject mail for non-existent mailboxes. *qmail*, by default, accepts mail for non-existent addresses. It determines later whether or not the user exists.

Unfortunately, this did not prevent spammers from getting connected to the mail server and getting messages into the mail queue. Many spammers make several parallel SMTP connections, so it was not uncommon to see 50 or more SMTP connections from a single IP address. Although the checklocal patch prevented messages to non-existent addresses from entering our mail queue, spammers essentially created a Denial of Service to the users of our mail server.

We immediately realized that the *qmail-uce* checklocal patch would not solve our problem on its own. Our next approach involved using *rblsmtpd*, which is part of the *ucspi-tcp* [10] package that includes *tcpserver* [11]. *rblsmtpd* attempts to block mail from RBL-listed sites by querying one or more RBL sources. *rblsmtpd* works with any smtpd server that runs under *tcpserver*. It can be configured to respond with a permanent (553) failure error message or a temporary (451) failure error message.

We attempted to implement *rblsmtpd* in several ways. First, we configured *rblsmtpd* to run continuously and deliver a temporary (451) error to RBL-listed sites. This prompted many customer complaints regarding legitimate mail not getting through. Next, we set up *rblsmtpd* to deliver a permanent (553) error to those RBL-listed sites. Again, this caused many of our customers to complain about mail bouncing.

Our last attempt involved running *rblsmtpd* only during times when we were being heavily spammed. Because it was configured to deliver a temporary error to RBL-listed sites, it was effectively useless. Spam would just queue up on the originating server until we turned off *rblsmtpd*. At that point, it was obvious that we needed something other than *rblsmtpd*. *rblsmtpd* did not help us solve the problem of our mail server getting pummeled by spammers' SMTP connections, and it brought on many complaints.

Some other alternatives to *rblsmtpd* include utilities such as Sieve [12] or SpamAssassin [13]. Unfortunately, these utilities must process each message individually. This results in a significant increase in the overall system resources required by the mail system. Compounded with the fact that spammers were already utilizing all of our server's resources, these utilities were not an option. Another option was to buy faster hardware or multiple servers. We opted for a homegrown software solution before investing in more hardware.

Design Goals

Each site is going to have its own unique problems when implementing an "out of the box" spam filter. In order to effectively implement a spam filtration system, a site needs to address these questions:

- What has not worked for us in the past?
- Do we have enough resources to allow client-side filtering options?
- Do we have the time and expertise to create our own spam blocking solutions?
- Would it be more effective to purchase faster and better hardware than to script a custom solution?
- How transparent does the spam blocking need to be to the user base?
- Are we concerned with bandwidth consumed by spam attacks?

After addressing the questions above, and ruling out failed approaches, we realized that we needed to refine our goals for a spam filtering system, and would probably need to engineer our own software-based solution based on those design goals.

Our requirements were:

- The method must conserve system resources.
- The method must reduce the amount of bandwidth consumed by spam attacks.
- The method must not add much additional overhead to mail processing.
- The method must prevent spamming sites from getting mail into the mail queue.
- The method must be manageable in a way that allows us to exempt certain hosts or networks.
- The method must keep our customers happy by minimizing the number of false positives.
- The method must be as transparent as possible to end users.

Our number one concern was to implement a solution which solved our problem without increasing the load on our mail server, as we did not desire a hardware upgrade at the time we were developing Deny-Spammers. This limitation ruled out utilizing a processor intensive spam control system, such as SpamAssassin or Sieve.

In addition to the headaches of a major mail migration, simply throwing hardware at the spam attacks would only be a short term solution. There has been, and will likely continue to be, a practically exponential growth in the amount of spam on the Internet. In a matter of months, our new hardware could be overwhelmed by additional and more creative spam attacks.

A hardware solution also fails to rein in the problem of bandwidth consumption by spam attacks. We simply wanted to reduce the ability of spammers to consume our resources (such as bandwidth and CPU utilization) conserving them for legitimate mail.

We choose to use frequency of attempts to deliver email messages to non-existent mailboxes as our heuristic. We later felt validated in choosing this metric because it was proposed by Jon Postel in RFC 706, which states, "A Host might make use of such a facility by measuring, per source, the number of undesired messages per unit time, if the measure exceeds a threshold, then the Host could issue the 'refuse message from Host X message...' " Other metrics, such as number of concurrent SMTP connections could be used to qualify the sending SMTP server as a spammer.

Once identified as a spammer, a method is needed to block future delivery attempts from that host to our mail server. We chose the most efficient method, which is to do the filtering at the IP level, in order to put the load of the filtration process on the operating system's kernel, rather than at the application level. Filtering at the application level, such as is done by SpamAssassin and Sieve, for example, would not prevent a spam Denial of Service attack.

Implementation/Solution

Deny-Spammers is a daemon that interfaces to the mail transfer agent and the firewall ruleset control program. It uses a strategy based upon patterns that spammers produce, including attempts to send to nonexistent addresses, to dynamically update the server's ingress rules. This approach moves blocking spam away from the delivery agent to the mail server's kernel, thus conserving system resources.

We chose to implement our filtration tool in Perl to increase the speed of development. This allowed us to have a working prototype within a few days of our idea's inception. Although the Perl implementation works well in production for our system, larger sites may want to consider using a more efficient development language. This becomes even more of a necessity as more heuristic tests are introduced.

We needed a solution that would work with the software that we were already using. Therefore, the current revision of Deny-Spammers is specifically designed for use on FreeBSD systems running *qmail*.

Presently, Deny-Spammers has two major prerequisites:

- any version of FreeBSD with IP firewall support installed in the kernel
- a patched version of *qmail* that includes the *qmail-uce* checklocal patch to reject mail for nonexistent addresses

Deny-Spammers interacts with the kernel's firewall by using FreeBSD's *ipfw* [14] program to ban and un-ban hosts. *ipfw* provides the user-level control of the firewall ruleset. Using Deny-Spammers with another operating system's firewall application would require additions to the code. Support for iptables, ipchains, packetfilter or IP Filter would all be simple to add. Modifying the system calls that Deny-Spammers makes would suffice to add support for any of these packet filters.

The types of patterns produced by spammers determines how Deny-Spammers interfaces to the mail transfer agent. In our case, spammers are detected by multiple sends to nonexistent addresses. In order to detect delivery attempts to nonexistent addresses in *qmail*, the *qmail-uce* checklocal patch was required. This patch provides a modification for qmail's SMTP daemon, qmail-smtpd, and logs details about each attempted delivery to a nonexistent mailbox, including the IP address of the host which attempted this delivery.

Although Deny-Spammers is currently dependent on the *qmail-uce* checklocal patch, it could be adapted to function with other MTAs. The exact implementation, would depend on the MTA itself. Sendmail, for example, defaults to logging delivery attempts to nonexistent addresses.

Deny-Spammers is intended to be executed at boot time and to run continuously. Many parameters can be fine-tuned from within the script. More information about these can be found in the code itself and in the implementation details that follow. To obtain the source code for Deny-Spammers, see the availability section near the end of this paper.

Because the checklocal patch logs all attempted deliveries to nonexistent addresses, Deny-Spammers monitors the system's mail log for messages emitted by the patch. These messages are parsed for the IP address of the host that attempted to make such a delivery. By defining thresholds of how many of these messages can be seen in a given time period, Deny-Spammers selectively prohibits hosts from making any more SMTP connections for a given "ban time" period.

To produce this behavior, three hash structures are used to track the state of the spam filtering system. One is used to track the times that hosts sent undeliverable messages, another for banned spammers, and another for the exception list.

The %spammer hash is a hash of lists. The keys of the hash are host IP addresses. The values of the hash are lists for each host which contains a list of timestamps. These timestamps represent times in which a host sent mail to a nonexistent address.

The %banned hash is a plain hash. The keys of the hash are the host's IP address. The values of the hash are scalars containing the timestamp in which that host was banned.

The %noban_list hash is a 4-level hash which contains the IP address exception list. This hash is organized such that the first level represents the first set of octets, the second level represents the second set of octets, and so on. The keys of this hash represent octets, with asterisks interpreted as wildcards. The values of the hash are '1' if a host or network is on the exception list.

The exception list is populated with the contents of the exception list file specified when the program starts. It is a flat ASCII text file containing one IP or network per line. An example exception list is provided with the distribution. The list is periodically reread by the script and any necessary firewall changes happen automatically.

There are a number of variables in the beginning of the script that may need to be adjusted based on the application. The most important parameters are the number of non-deliverable message attempts during a time span that occur and the time span itself. Ten attempts during a five-minute period has worked for us. The ban time, or length of time a host stays banned, is also adjustable. Lower ban times will typically keep the size of the firewall ruleset reasonably small.

Other variables include the path to the exception list, log files, *ipfw* and the regular expression used to match incoming timestamps and IP addresses from the mail log input. With the *qmail-uce checklocal* patch, nonexistent user messages should appear in the system's mail log as shown in Listing 2.

Periodically the program will prune the banned list, unbanning hosts which have been banned for the length of time specified by the administrator or hosts which have been added to the exception list since the last refresh. The refresh time is also configurable.

An end rule is also defined. If this rule number is ever reached, Deny-Spammers will clear the existing ruleset and start over. This feature is useful if a server can only handle a certain number of rules efficiently.

The pseudocode in Listing 1 shows the infinite loop which occurs right after initialization. It is where virtually all of the processing occurs in Deny-Spammers. Incoming IP addresses from the mail log lines are matched against a regular expression and are parsed. The IP address and timestamps are stored for each nonexistent user message. Every time a line is received and tracked, the program decides if the IP should be banned based on the given parameters in the code.

We have been using the same algorithm since Deny-Spammers was put into production. Only minor changes to the parameters and bug fixes have been required to produce the results we desired. For example, we keep the firewall ruleset small by using a relatively short ban time of three days, as opposed to, say, two weeks.

We have only implemented one spam signature pattern so far. Other metrics could be developed and implemented in a similar manner as described above. In most settings, each test would have to be carefully chosen, designed, and assessed for minimal negative impact to the users. Different tests are likely to intrinsically block more legitimate mail than other tests. If multiple signatures were available, sites may also want to pick and choose depending on the needs of their

```
While (true) {
    Match incoming lines against a regular expression for
    undeliverable messages to nonexistent addresses and parse
    timestamp and IP address.
    Skip line if host is in the exception list.
    Trim the timestamp list for this host to $MAX_SPAMMER_ENTRIES.
    Add the timestamp to the host's list contained in the spammer
    hash.
    Check how many delivery attempts to nonexistent address this host
    has made in the sampling interval, $SPAM_TIMESPAN.
    If nondeliverable messages > $SPAM_TRIGGER then filter this IP.
    If current time >= $next_refresh then
        calculate next refresh,
        reload the exception list and prune the banned hosts list
        (un-ban hosts who have been banned for $BAN_TIME)
}
```

Listing 1: Pseudocode for infinite loop.

```
Jan 1 00:00:00 mailhost smtpd: 1234567890.123456 12345: DENYMAIL:
RCPT_TO:_Filter.NoUser:_ relay unknown [123.123.123.123] FROM
<bounce@your-info.net> ADDR <abcdefgh@telerama.com>
```

Listing 2: Typical nonexistent-user message.

users. Fine-tuning the tests may also be required, again depending on their needs, to achieve the desired results.

In Production

Figure 1 shows the number of attempted deliveries to non-existent mailboxes and the number of firewall rules over a four-day period. The first two days shows Deny-Spammers running. For graphing purposes we intentionally reset it three times. The last two days show what happens when Deny-Spammers is disabled.

A five minute sampling interval was used for both the number of firewall rules as well as the number of undeliverable mail attempts.

The graph starts with one firewall rule on April 25th. At this point Deny-Spammers was initialized. At this point, the firewall rules begin to increase. As the firewall rules approach 1,000, the delivery attempts average around 100-200 attempts every five minutes.

When the firewall is reset, the firewall rules go back to one, and the delivery attempts begin to increase. As the delivery attempts decrease, the firewall ruleset starts increasing at a slower rate. Notice that the firewall rules increase very quickly when the number of delivery attempts counts is high and the firewall has just been reset.

Shortly after midnight on April 27th, the firewall is at 1 for the third time. The delivery attempts increased dramatically when Deny-Spammers was disabled for about six hours.

Deny-Spammers was restarted where the firewall rules start to increase again (on the morning of April 27th). A similar pattern occurs, the delivery attempts decrease as the firewall rules increase.

In the last two days of the graph, Deny-Spammers was completely disabled. The delivery attempts average more than twice that of when the spam filtration system was enabled.

Limitations

Despite its usefulness, Deny-Spammers has some limitations.

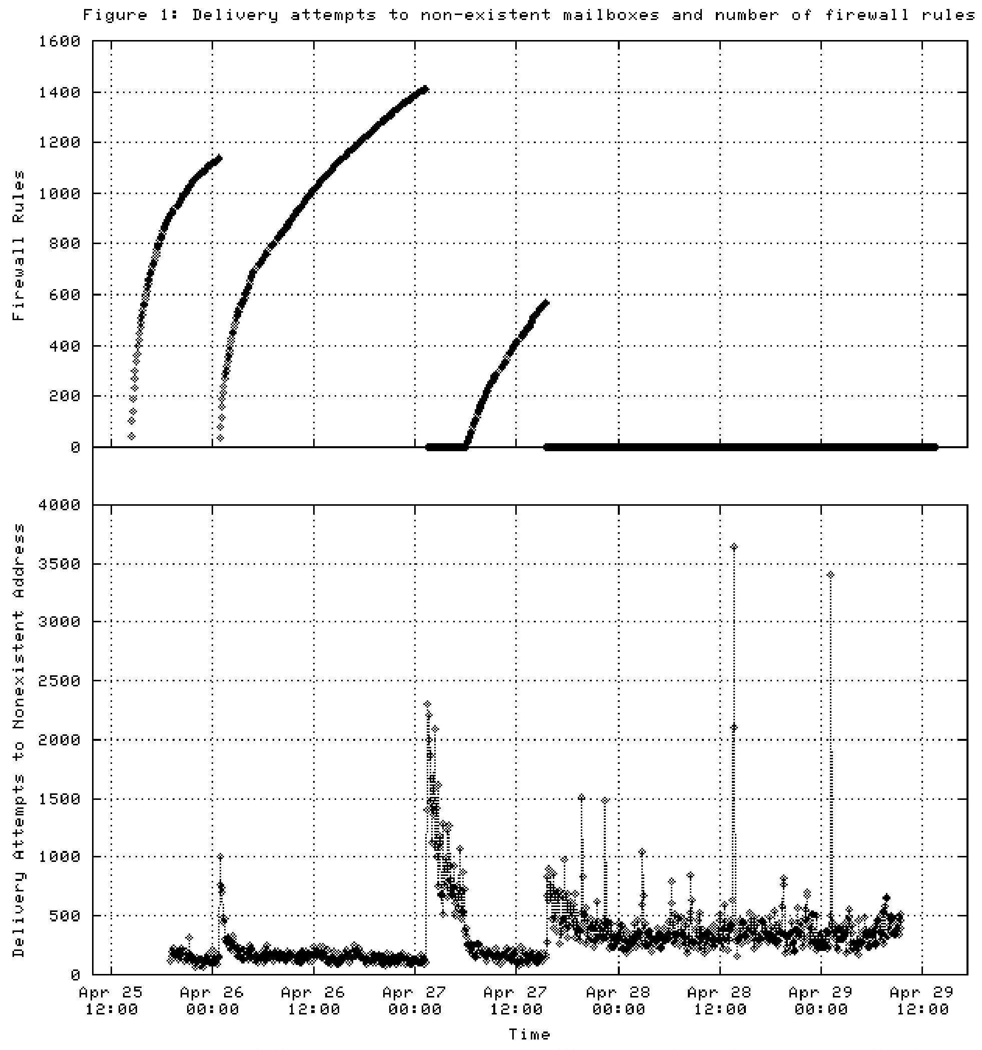

Figure 1: Attempt deliveries to non-existent mailboxes and number of firewall rules.

- The exemption list only supports individual hosts and/or classful networks. CIDR notation is not supported.
- Deny-Spammers is currently only compatible with FreeBSD machines running *qmail*.
- Another known issue with the software is that qmail-smtpd processes will hang for a while if a host is banned while they have an active connection. This is mostly a problem when the script is first started as the initial surge of banning causes a large number of such hung processes.
- Scalability is limited in some cases. The kernel firewall code itself may not be able to efficiently process thousands of rules, which is a common scenario. We have determined that older revisions of FreeBSD, for example, are subject to this problem. With older versions, firewall rules were stored in a linked list. Newer revisions use a tree structure, making the handling of large rulesets much more efficient. Using an inefficient data structure ultimately caps the maximum number of rules that a server can handle efficiently.
- Spammers could exploit the *qmail-uce* checklocal patch to find valid addresses. *qmail*, by default, doesn't allow a sender to know whether or not an address is valid. If this was a major concern, the checklocal patch could be modified so that it only logs the attempts, instead of logging and bouncing the message. In this case, spammers wouldn't be able to verify addresses, and Deny-Spammers would still operate correctly.

These limitations have not prevented Deny-Spammers from being a very useful tool. Customers are not receiving as much spam and it effectively deals with Denial of Service instances caused by spam. Customer complaints have been minimal compared to our other approaches. When a false positive is detected by a user and brought to our attention, simply adding the correct host or network to the exception list should be all that is required to resolve the issue.

Future Plans

Deny-Spammers was developed as a quick and dirty hack to solve a pressing problem that we had. As such, it works very well for us, on our platform, and for our staff, who understand its limitations.

Now that it has been in production for over a year, and providing the results we desire, we can see many areas in which to expand its functionality to make it attractive and usable for the general community. Some obvious improvements planned are addressing Deny-Spammer's current lack of scalability and interoperability, and adding a GUI interface to allow non-administrators to access its log files and exception list.

- Scalability issues. Implement Deny-Spammers for a mail server farm or a single server with many more users.
- Add the ability to use a separate firewall. Currently, the firewall must be located on the same machine that is processing mail. A separate firewall could be updated remotely via an ssh tunnel. This feature could also be useful when scaling this application to a mail server farm, so that one firewall could be responsible for a group of mail servers. Given a secure communication mechanism to update the firewall rules, such an improvement should be straightforward.
- Integration with third-party applications such as SpamAssassin or Anomy Sanitizer. Allow results from SpamAssassin/Anomy to determine whether or not a host gets banned.
- Improve statistical generation for research purposes. Create historical averages of number of hosted blocked over long periods of time. Look for interesting patterns, such as whether spam comes in bursts and when it most frequently occurs.
- Develop a better interface for unbanning hosts and managing the exception list. Add CIDR notation support for the exception list.
- Interoperability with other operating systems. This is simply a matter of adapting the firewall system calls to work with various firewall implementations (ipchains, iptables, IP Filter, packetfilter).
- Interoperability with other mail transfer agents (sendmail, postfix, et cetera). Because each program works slightly differently, interfacing each spam signature pattern could be a tedious process.
- Develop more "spam signatures." A few other patterns we've considered using as criteria are:
  1. The number of concurrent SMTP connections made by a host – experience has shown that spammers are capable of making many parallel SMTP connections to the same mail server.
  2. The number of recipients a message is sent to – Spammers often send messages with extremely large RCPT TO lists.
- A point system for hosts could be introduced, such that multiple spam signature patterns are taken into account for each host (similar to the 'hits' mechanism used by SpamAssassin).

Availability

Deny-Spammers is freely available source code and documentation can be found at http://deny-spammers.telerama.com. Deny-Spammers is written in Perl 5 and developed in and tested under FreeBSD. It contains no dependencies on non-standard modules or libraries. Specific questions regarding Deny-Spammers can be sent to denyspam@telerama.com.

Conclusions

This paper describes a stateful inspection strategy for dynamically creating firewall rules that block access from mail hosts based upon their recent behavior. If the sending mail host is determined to be a spammer (based on our criteria) a daemon updates the firewall ruleset for our mail server.

Proving the success of our strategy is difficult due to the impossibility of measuring the lack of an event (lack of delivery of spam messages). Many alternative approaches were too resource-intensive for us to implement. We found that other spam filters that were acceptable for us resource-wise (such as MAPS-RBL) created large and obvious negative customer feedback. We have not had that backlash upon implementing Deny-Spammers. We feel that we have fewer customer complaints about receiving spam, but we don't have enough data to support that point empirically.

What our data does show is that we are banning thousands of misbehaving mail hosts based on our metrics. We believe all of these hosts to be likely spammers.

Authors

All of the authors have worked at Telerama Internet (http://www.telerama.com) for the past several years.

Deeann M. M. Mikula is the Director of Operations and Junior Unix System Administrator at Telerama Internet, and is co-founder of the local SAGE Chapter in Pittsburgh. She has worked as a Behavioral Neuroscience Researcher, a Coffeehouse Manager and a Visual Artist. When not in front of a keyboard, she can be found drinking scotch at a Gothic club or painting and drawing. Deeann can be reached via email at deeann@telerama.com.

Chris Tracy is Telerama Internet's Senior Network and Systems Engineer and a SCinet Volunteer. He holds a Bachelor's Degree in Computer Engineering from the University of Pittsburgh. When not in front of a keyboard, Chris can be found drinking beer, DJ'ing or playing drums. Chris can be reached via email at chris@telerama.com.

Mike Holling is a part-time Network Engineer with Telerama Internet. He holds Bachelor's Degrees in Computer Science and Electrical Engineering from Carnegie-Mellon University. When not snow boarding, skate boarding or drinking beer, Mike can be found working as a Network Consultant in Whitefish, Montana. Mike can be reached at myke@telerama.com.

Acknowledgments

We are grateful to many people for their contributions to this project and this paper. We would like to thank Doug Luce, owner and CEO of Telerama, for fostering a workplace where we are encouraged to try novel approaches to problems. We thank our fellow staff at Telerama for putting up with the real-time tweaking of our production mail server.

Especially valuable in producing this paper was peer review and support. We would like to thank Esther Filderman and Josh Simon for encouraging us to publish our work, and for support along the way. The advice of our shepherd, John Sellens, and the comments of our anonymous reviewers, were invaluable in shaping our final paper.

References

[1] Postel, Jon, "RFC 706: On the Junk Mail Problem," November 1975. Network Working Group. 10 April 2002, http://www.faqs.org/rfcs/rfc706.html.

[2] Lee, Jennifer B., "Spam: An Escalating Attack of the Clones," *New York Times*, June 27, 2002.

[3] Gomes, Lee, "How Hotmail Keeps Its Email Empire From Spam's Clutches," *Wall Street Journal*, July 8, 2002.

[4] Gartner Consulting, "ISPs and Spam: The Impact of Spam on Customer Retention and Acquisition," June 14, 1999.

[5] "Mail Abuse Prevention System LLC," http://mail-abuse.org/rbl.

[6] Bernstein, Dan, "*qmail* home page," http://www.qmail.org.

[7] Varshavchik, Sam, "MAIL 1.01 unified Anti-UCE/Mailbombing patch," http://portofhoodsport.org/qmail/misc/uce.html.

[8] Postel, Jonathan B, "RFC 821: Simple Mail Transfer Protocol," August 1982. Information Sciences Institute: University of Southern California, 20 April 2002, http://www.ietf.org/rfc/rfc0821.txt.

[9] Bernstein, Dan, "The *rblsmtpd* program," http://cr.yp.to/ucspi-tcp/rblsmtpd.html.

[10] Bernstein, Dan, "*ucspi-tcp* home page," http://cr.yp.to/ucspi-tcp.html.

[11] Bernstein, Dan, "The *tcpserver* program," http://cr.yp.to/ucspi-tcp/tcpserver.html.

[12] Showalter, T., "Sieve: A Mail Filtering Language," January 2001. Network Working Group, 14 April 2002, http://www.ietf.org/rfc/rfc3028.txt.

[13] Hughes, Craig R., "SpamAssassin home page," http://www.spamassassin.org.

[14] Ugen J. S. Antsilevich, Poul-Henning Kamp, Alex Nash, Archie Cobbs, and Luigi Rizzo, "Manual page for *ipfw* – IP firewall and traffic shaper control program," http://www.freebsd.org/cgi/man.cgi?query=ipfw.