A Better E-Mail Bouncer

Richard J. Holland – Rockwell Collins, Inc.

ABSTRACT

This paper describes a portable electronic mail bouncer which sends detailed information back to the sender when a mail message can not be delivered to it's intended recipient. The bouncer was originally written to handle a large merger between multiple DNS domains, and is implemented entirely in Perl5 as a mail delivery agent. The bouncer operates under the concept of "least privilege" so it's safe to run directly from mail transport agents such as sendmail. The bouncer is designed to make the human processes and interactions in dealing with undeliverable E-mail easier for both postmasters and end-users alike.

Introduction

Local Background

Most large networks are in a constant state of flux when it comes to account management and electronic mail routing. New users are added daily as old users are removed, often without thought of potential future consequences in an ever expanding electronic world. As more and more computer neophytes start using the Internet, handling mail delivery problems and bounces will become a much larger problem for any large site's postmaster.

Like postmasters at most sites with several thousand users, our bounced mail is run through multiple filters in an attempt to either auto-respond to problems or to sort the problems into related issues for easier handling. The rapid increase in the amount of unsolicited commercial E-mail is making this even more important. This paper examines the other side of these issues – what an end-user sees in an undeliverable message, rather than what a postmaster sees. Hopefully by improving the end-user interface, we will lower the number of undeliverable messages a postmaster must deal with directly.

This spring, Collins Avionics & Communication Division merged with it's sister company, Collins Commercial Avionics, to form Rockwell Collins, Inc. The new combined enterprise network contains over 10,000 active network users, and several thousand old accounts which no longer exist, but whose userid's can not be re-used in an effort to prevent mis-delivered electronic mail. One of the first steps in merging networks of this size are to correct any namespace collisions, both hostnames and login IDs. Forcing uniqueness of existing usernames generally isn't looked upon favorably by the person whose address is changing; they may have their old address printed on business correspondence and recorded in thousands of "From:" headers scattered across the Internet. In an enterprise network, even a collision rate of less than 5% can make a user-friendly solution a value-added task. In an effort to help our users and customers do business more efficiently, we're notifying senders when an address changes with a clearly written explanation of what happened and why. This bounced message gives the sender the recipient's new address, similar to the sendmail redirect [1] feature, but with more detailed information.

Why E-Mail is Bounced

In most cases it is possible to use alias maps on mail hubs and gateways to re-route electronic mail to the user's new address automatically, but usually the user has no way to control how long this forwarding is enabled. When it is removed, the old address suddenly generates undeliverable messages with brief errors such as those shown in Listing 1. While this tells the user why the message was returned, it doesn't explain why the username they're sending to is now unknown,

```
... while talking to mailhost.domain.com.:
>>> RCPT To:<user@mailhost.domain.com>
<<< 550 <user@mailhost.domain.com>... User unknown
550 user@mailhost.domain.com... User unknown
```

Listing 1: Undeliverable message error.

```
... while talking to mailhost.domain.com.:
>>> RCPT To:<user@mailhost.domain.com>
<<< 551 User not local; please try <user@elsewhere.com>
551 User not local; please try <user@elsewhere.com>
```

Listing 2: Forwarding address in return message.

nor does it suggest any corrective actions. While the above example makes perfect sense to any Postmaster or System Administrator, many end-users simply don't understand how to interpret many computer-generated error messages like this, and then must contact a System Administrator or attempt to contact the person they're trying to send the E-mail to via alternate means to get the new E-mail address.

Newer versions of sendmail (Berkeley v6.25 and later) can be configured to take advantage of a feature called redirect which is used to provide forwarding addresses in the returned message, rather than just saying "User unknown." A sample redirect bounce might look like Listing 2. Once again sufficient for System Administrators, this time the error message communicates the recipient's new address. However, these terse error messages often confuse end users who may wonder if this is just informational and whether or not their message was delivered to the new address provided or not.

Bouncer Design Considerations

With a user community of over 10,000 people, we wanted to develop a methodology for returning undeliverable messages to senders with concise, plain English explanations of why the message is being returned, and suggest corrective actions they should take to ensure their message is delivered successfully in the future. The bouncer needs to handle many potential reasons for a userid change such as namespace collisions, legal name changes, employees who have left the company, etc. In order to facilitate rapid communications, in addition to returning a mail message to the sender, where possible an attempt is made to deliver the message to the user's new address if permitted by local security policies.

The final implementation language was also given careful consideration. First and foremost, because of our migration schedules we needed a language which would facilitate rapid prototyping and development. Ideally, the language would also allow us to easily secure the bouncer. Perl [2] meets both of these requirements for several reasons:

- Perl scripts are usually shorter than comparable shell scripts or C programs, and are therefore easier to maintain. The entire bouncer program is under 1,000 lines of code, including comments.
- Most functions the bouncer needed to perform could be handled with native perl functions, which means we didn't need to pass user input to a shell, with potentially disastrous results. We could also take advantage of Perl's *taintperl* program, which considers any user-supplied data as "tainted" and therefore unsafe for shell interpretation.
- Perl's unmatched regular expression capabilities were the final deciding point. Since a good portion of the bouncer's task is to parse text (configuration files and E-mail messages) and perform actions based on recipient and sender information, Perl was a natural choice for a fast implementation which is easy to maintain by even junior System Administrators.

Implementation & Configuration

Filter Implementation

The initial implementations of the bouncer program ran with minimal privileges as a simple mail filter. Users who were to have their mail bounced would receive an alias similar to:

```
oldname: newname,bouncer or
oldname: bouncer
```

In this way, the recipient would receive a copy of the message if permitted by local security restrictions, and the bouncer account would also receive a copy. The bouncer account contained a simple .forward file which reset IFS for security purposes, executed the bouncer script, and if it failed, returned an exit status of 75 so that sendmail would bounce the message as it normally would have. The .forward file read:

```
|IFS=' ' & exec /path/bouncer.pl\
    || exit 75 #bouncer
```

The first problem with this implementation is that mailing list distribution programs write their headers in many different ways. There doesn't seem to be any standard as to which header will contain the mailing list address, and which will contain the mailing list maintainer's address.

The other problem with the filter implementation is that if the original recipient appears only in a Bcc: header, the recipient is hidden from the filter; only the delivery agent knows who to deliver the message to. By the time the message reaches the bouncer, the Bcc: headers have been removed for privacy reasons by the mail transfer or delivery agent.

Delivery Agent Implementation

Because of these problems, the bouncer was rewritten as a mail delivery agent. This required the addition of a few lines to the mail system's sendmail.cf configuration file, and a re-write of the bouncer code for security reasons. At the same time, the code was moved to the mail server's local file systems, rather than the bouncer account's home directory (in fact, the bouncer account is no longer needed; the delivery agent can be run as any unprivileged user, such as "nobody"). Because the bouncer is running as a delivery agent, it must run under the same assumptions a SUID [3] program would run, since it's launched via sendmail with system privileges. Since the bouncer doesn't actually have to do any local mail delivery, it can relinquish these privileges immediately (running instead as "nobody"), which minimizes the risk of security issues. In addition, it carefully "untaints" all user input that is passed back to external programs, to prevent a shell from interpreting malicious data.

Configuration of the delivery agent requires the addition of two lines to the sendmail.cf file. The first addition is a mail delivery agent command line, such as:

```
Mbouncer, P=/bin/bouncer,
     F=DFMPlms S=10, R=20,
     A=bouncer -a $f -d $u
```

This line fully defines the operation of the bouncer program and it's interaction with sendmail. The F= flag tells sendmail that the bouncer needs a Date: header (D), a From: header (F), a Message-ID: header (M), and a Return-Path: header (P). It also states that /bin/bouncer is a local delivery agent (l), that multiple recipients are permissible (m), and to strip quotation marks (s). The S= flag tells sendmail to process the sender's envelope and header addresses with ruleset 10 (after ruleset 3 and 1, but before ruleset 4). The R= flag tells sendmail to process the recipient's envelope and header addresses with ruleset 20 (after rulesets 3, 0, and 2, but before ruleset 4). The A= flag declares the command line arguments the bouncer will see in it's argv[] array. In this case, sendmail will pass the sender's envelope address after the -a option ($f macro) and will pass multiple recipients after the -d option ($u macro).

Once the delivery agent has been established, ruleset 0 must be modified in order to enable it. If addresses of the form *user@bouncer* are to be handled, the line in Listing 3 is added to ruleset 0. If addresses of the form *user@newdomain.com.bouncer* are to be handled, the line in Listing 4 is added to ruleset 0. Which of these rules are used is at the discretion of the postmaster implementing the bouncer; either or both (or other similar configurations) can be used, depending on the local site's configuration preferences. For example, if there is already a host named bouncer in the new domain, the second address specification would be preferable to avoid confusion. The bouncer program will handle both cases, as ignores everything after the username portion of the address.

Functionality & Configuration

The bouncer code has several configurable options within the code itself. These options are all defined at the beginning of the program and allow the administrator to set:

- Who receives E-mail/pages if the bouncer fails in some unforeseen way?
- Who should the bouncer deliver mail as (e.g., MAILER-DAEMON, postmaster, etc.)?
- Who should replies to bounced messages be delivered (e.g., helpdesk, postmaster, etc.)?
- Which Precedence: headers indicate a message that should not be bounced?
- If a message isn't bounced, is it delivered with an addendum to the original recipient?
- Who should the bouncer run as (i.e., "nobody" or another specific user)?
- Error messages & explanatory text settings for various situations.

The address configuration file is relatively straight forward and allows the administrator to specify the old address, new address, some personal contact information for the recipient, and a configuration code used to determine which explanatory text is sent in the bounced messages for each user.

In an attempt to prevent mail loops, the bouncer follows RFC 822 [4] conventions for returning undeliverable mail. Additionally, in order to deal with mailing list programs which put the list maintainer's address in the Errors-To: header, it is given precedence. If there is no Error-To: header, the Sender: header is used to determine how to send the bounce to. In the absence of a Sender: header, the From: header will be used, and if the From: header does not exist, a last ditch effort is made using the sender's address from the SMTP message envelope.

Finally, any message with a Precedence: header matching an administrator-configurable setting will be handled by sending a separate not to the recipient, rather than replying to the sender. In this way, users will be reminded to update their mailing list subscriptions with their new address, and there is a much smaller risk of starting mail loops between the bouncer and a mailing list program. By default, Precedence: headers which trigger this behavior are any that match *bulk*, *junk*, *list* [5].

Logging Bounced Mail

Logging of all bounced mail is handled by sendmail itself, which will create an entry in the syslog output just as if the normal local delivery agent had been run. In this case however, the local delivery agent will be the bouncer, so it's relatively easy to pull a list of all bounced E-mail messages from the syslog output. A sample syslog entry is shown in Listing 5.

Sample Configuration

For a simple example of how the bouncer is configured, assume there is someone named John D Smith on one network, and another user named Jeff D Smith

```
R$+<@bouncer>  $#bouncer $:$1<@bouncer>     user@bouncer
```

Listing 3: Ruleset 0 rule to redirect bouncer messages

```
R$+<@$+.bouncer> $#bouncer $:$1<@$2.bouncer> user@*.domain.com.bouncer
```

Listing 4: Additional rule for *.bouncer

on a second network. John's E-mail address is *jdsmith@domain1.com* and Jeff's address is *jdsmith@domain2.com*. When the two networks are combined, it is desirable for each to answer for the new network (domain3.com) as well as both old networks (domain1.com and domain2.com) for backward compatibility. In this manner the same mail servers may be used to provide redundant delivery hubs for the same domain.

In order to implement this improved delivery system, one or both of the addresses above much change. Sendmail's *redirect* can't be used in this case because it only handles the case where one user changes their address; in this case, both users should change their addresses to avoid confusion, since different people may remember either Jeff or John as *jdsmith@[someplace].com*. If only Jeff were to change his userid, then John would likely start receiving mail in the future which the sender actually intended for Jeff, not realizing that Jeff had changed his userid.

The solution in this case is to change both addresses to something like *jdsmith1@newdomain.com* for John, and *jdsmith2@newdomain.com* for Jeff. In this way, both are unique and neither maintains the original address. Prior to changing the userid's, the bouncer is configured to respond to any of the following addresses:

```
jdsmith@domain1.com
jdsmith@domain2.com
jdsmith@newdomain.com
```

with a message that might look like that in Figure 1. This response would expedite the updating of address information that an external customer might be maintaining, while at the same time preventing phone calls to John or Jeff. It will also hopefully prevent the end users from having to contact a Network Helpdesk or the local postmaster to inquire why mail to jdsmith generates a 'username unknown...' message when it was successfully deliverable in the past.

The configuration of the bouncer program's reply is contained within a single ASCII file. The format of the file is:

```
code=user1,user2,[...]: \
   text (address1@[domain]), \
   text (address2@[domain]),[...]
```

where code is a two digit numeric code, user# is a username, text is any arbitrary text, and address# is the corresponding new address for user#. The domain is optional, and if left off, a default domain will be inserted automatically (configurable via an option in the bouncer code). By grouping multiple users on the same line or multiple lines using a backslash (\) as a line continuation character, the names and addresses displayed for a given address can be controlled. In the scenario for John and Jeff given above, the bouncer address configuration would look like:

```
01=jdsmith: John D Smith \
    (555-1212) (jdsmith1@), \
    Jeff D Smith (555-1234) \
    (jdsmith2@)
```

```
Sep 6 10:57:35 mailhub sendmail[527]: AA186621431: from=<user@somedomain.com>
Sep 6 10:57:35 mailhub sendmail[527]: AA186621431: size=80, class=0,
    pri=10080, nrcpts=2
Sep 6 10:57:35 mailhub sendmail[527]: AA186621431:
    msgid=<9708068735.AA873561403@somedomain.com>
Sep 6 10:57:35 mailhub sendmail[527]: AA186621431: relay=somehost [127.0.0.1]
Sep 6 10:57:36 mailhub sendmail[527]: AA186621431: to=jdsmith@bouncer,
    delay=00:00:01, stat=Sent, mailer=bouncer
```

Listing 5: Sample syslog entry.

```
Due to the consolidation of networks between domain1.com and
domain2.com to form a common network newdomain.com, some Electronic
Mail addresses were changed. This message is automatically generated
in response to your E-mail to one of the persons listed below. In
this case, we were unable to determine which user you intended to
send to, and request that you re-send your E-mail message to the
new address listed after the person's name below:

 John D Smith (Phone Number)   [jdsmith1@newdomain.com]
 Jeff D Smith (Phone Number)   [jdsmith2@newdomain.com]

For your convenience, a copy of your original message is included below.
=====================================================================
[original message including headers]
```

Figure 1: Sample of a bounced message with indeterminate recipient.

Since no domain is specified, newdomain.com is appended automatically. This makes it simple to update if one of the jdsmith's moves into yet another domain later (i.e., leaves the company, changes departments, etc). It could then be updated to read:

```
01=jdsmith: John D Smith \
    (555-1212) (jdsmith1@), \
    Jeff D Smith (555-9875) \
    (jdsmith2@elsewhere.com)
```

The numeric code (01 in this example) is used by the bouncer program to determine which explanation is prepended to the bounced message. For example, you might set up the following codes and explanations:

- Code "01" to be used for a division merger with explanation of: Due to the consolidation of networks between domain1.com and domain2.com to form a common network newdomain.com, some Electronic Mail addresses were changed. This message is automatically generated in response to your E-mail to one of the persons listed below. In this case, we were unable to determine which user you intended to send to, and request that you re-send your E-mail message to the new address listed after the person's name below:
- Code "02" to be used for name changes with explanation of: The person you sent mail to has changed their mail address because their name changed. This message is automatically generated in response to your E-mail, and has been delivered to the new mail address for you. This is only an information message to allow you to update your records to reflect the user's new address.
- Code "03" may be used for users who have left the domain: The person you sent mail to listed below no longer has a mail address on this network. For security reasons, your message has not been forwarded to the person. Please resend your message to their new address listed below if you still would like them to receive it.

A fully populated configuration file might then look like that shown in Figure 2. Once the bouncer is ready to handle the invalid addresses, the aliases file would be updated accordingly, as in Figure 3.

Alternative Solutions and Future

Alternative Solutions

As currently written, the bouncer behaves in a similar manner to the *vacation* [6] program. However, the bouncer is centrally managed and has finer control over what happens with each recipient's messages. The only configuration which needs to be done is to initially set the administrator options in the bouncer code, update the address information inside the single ASCII address configuration file, and configure the aliases for old usernames so they will be directed at the bouncer. If *vacation* were to be used for this implementation, each old account would need to be maintained, have a .forward file, and an individual *vacation* configuration. In the sample case provided earlier, *vacation* would not be a viable long-term solution when the password maps between domain1 and domain2 are eventually merged, because of the username conflict. While this problem can be circumvented with the creative use of aliases and sendmail.cf rules, the configuration of the bouncer is much simpler and obviates the need to maintain multiple configurations for each account that's changed.

Another alternative would be to take advantage of sendmail's *#error* delivery agent. However, this would require further modification of the sendmail.cf file in order to provide multiple error conditions, and tends to produce terse error messages. This would not meet the design goals of the bouncer – simple configuration and detailed explanations – and is more difficult

```
# Users who've had their address changed because of a namespace
# collision in the domain merger
01=jdsmith: John D Smith (555-1212) (jdsmith1@),\
    Jeff D Smith (555-1234) (jdsmith2@)

# Users who've had their address changed because of a name change
02=jadoe: Jane A Smith (formerly Jane A Doe, 555-1111) (jasmith@)

# Users who're no longer valid on this network
03=jsbrown: John S Brown (moved to Div XYZ) (jsbrown@xyz.domain.com)
```

Figure 2: Fully populated sample bouncer.cfg file.

```
jdsmith: jdsmith@bouncer      # can't tell who they want to send to here;
                              # just bounce it
jadoe: jasmith,jadoe@bouncer  # forward and notify sender of new address
jsbrown: jsbrown@bouncer      # dont fwd -- may have confidential
                              # information: just bounce it
```

Figure 3: Sample mail.aliases map for sample bouncer.cfg configuration.

to maintain, especially by junior administrators. The advantage to the bouncer is that once the sendmail.cf is initially set up, no further modifications need be made. Junior System Administrators can modify the bouncer.cfg and mail.aliases maps to configure new bouncer entries.

Future Direction

If enough traffic is being directed through the bouncer, it might be better implement in a compiled language such as C, to prevent the startup costs of Perl. However, the current delivery agent implementation takes less than 1 second to execute, so it would take a very large site to justify this more complex task. For example, our site contains over 10,000 accounts. If a namespace collision rate of 5% is assumed, this means that 500 usernames are going to be handled by the bouncer. If each address receives 10 pieces of E-mail per day, this is only 5,000 messages to handle. Most modern gateways can handle much more than this without any performance problems in an average day; ours typically process 75,000 to 100,000 messages in an average day with no noticeable performance problems. With the load balancing provided by MX record preferences [7], only extremely large sites may ever need to re-implement the bouncer for performance reasons.

The main feature lacking from the bouncer is the ability to auto-detect and stop mail loops. As much care as possible was invested in determining sender information and handling mailing list processors, but there is still small possibility that the bouncer program could get into a loop with another mail delivery agent such as a mailing list processor. While this shouldn't happen with such popular mailing list packages such as majordomo or listserv, not everyone uses these to process heir mailing lists, and not all mailing list processors follow the RFC's and use the correct header formats. An algorithm for detecting and preventing these types of mail loops will be added to future releases of the bouncer code.

Availability

Please contact the author directly for further information and program availability information.

Author Information

Rich Holland is the Technical Lead for the Enterprise E-mail Team at Rockwell Collins, Inc. where he is responsible for technical leadership and future direction in merging multiple mail systems into a common system for use by over 10,000 end users worldwide. Before coming to Rockwell, he was a Senior System Administrator for Synopsys where he oversaw the care and feeding of the Synopsys Porting Center machines. Reach him via U.S. Mail at Rockwell Collins, Inc.; M/S 106-193; 400 Collins Rd. NE; Cedar Rapids, IA 52498. His E-mail addresses are <rjhollan@collins.rockwell.com> and <holland@pobox.com>.

References

[1] *Sendmail 8.x (Berkeley) Release Notes*, ftp://ftp.cs.berkeley.edu/pub/sendmail.

[2] *Programming Perl*, Randal Schwartz and Larry Wall, O'Reilly & Associates, Inc., 1991.

[3] *Practical UNIX Security*, 2nd Ed., Simson Garfinkel and Gene Spafford, O'Reilly & Associates, Inc., April 1996.

[4] *RFC 822: Standard for the Format of ARPA Internet Text Messages*, Network Information Center, 1982.

[5] *Sendmail*, Brian Costales, Eric Allman, and Neil Rickert, O'Reilly & Associates, Inc., November 1993.

[6] *vacation*(1) man pages.

[7] *DNS and BIND*, 2nd ed., Paul Albitz, O'Reilly & Associates, Inc., January, 1997.

[8] Perl Home Page, http://www.perl.org/ .

[9] AUSCERT Security Papers, http://www.auscert.org.au/information/papers.html .