The following paper was originally published in the
Proceedings of the Eleventh Systems Administration Conference (LISA '97)
San Diego, California, October 1997

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org

Selectively Rejecting SPAM Using Sendmail

Robert Harker – Harker Systems

ABSTRACT

With the growing popularity of the Internet, unsolicited electronic mail (spam) has become a major concern. It fills up user's mailboxes, clogs mail relays, wastes postmaster time, and creates ill will for sites that have been used as a relay. Most sites want to filter spam before they receive it but filtering spam is hard to do without filtering legitimate mail messages.

This paper discusses what characterizes spam and describes rulesets that can be added to a *sendmail* version 8.8 [1] *sendmail.cf* file to selectively reject mail from specific addresses, domains or IP addresses and to prevent spammers from relaying mail through a site. It discusses the different issues facing corporate sites and the special issues facing Internet Service Providers (ISP). The rulesets presented have been implemented as *M4* template files so they can be easily integrated into a sendmail 8.8 sendmail.cf file as a FEATURE using M4. These rulesets are currently in use at Harker Systems and other sites, and are available via anonymous ftp.

Motivation

Unsolicited email, or spam, has become a chronic problem on the Internet. It fills user's mailboxes and clogs SMTP mail relays. It creates ill will and wastes postmasters' time. Filtering spam is a ongoing challenge.

It is easy enough to filter out spam which has valid addresses from a known spam site. Regrettably, spammers are very creative in how they disguise their messages. Also, one person's spam is another person's interesting junk email. And, if a non-spam message is sent from an address which is defined to be a spam address, it will be rejected.

Because of these issues, filtering spam must be handled with care. It is as much a policy and procedure problem as it is a technical problem. Creating the tools to filter spam is relatively easy. Creating the policy and procedure of what constitutes spam at your site, and what to block, is much more problematic.

One of the important new features of sendmail version 8.8, is a new set of four rulesets which are used to reject mail messages. The check_, *check_relay, check_mail, and check_rcpt set of rulesets allow rejection of a message during SMTP reception – before it is transferred to the destination or a relay. This off-loads the processing and generation of a bounce message by the mail host or SMTP relay. An additional ruleset, check_compat, can used to reject mail during sendmail's delivery of a message after it has been accepted.

Implementation of these rulesets is evolving with versions developed by Eric Allman [2], Claus Assmann [3]. The results presented in this paper represent a next step in this evolution. My students' questions caused me to think about what characterized spam and to add additional tests to the rulesets. The spammers courteously provided copious quantities of sample spam messages so I could test my assumptions.

What Is Spam?

The simple answer is that spam is any unsolicited email that is sent to a large list of users without their prior permission. When someone receives a piece of spam, they recognize it as such and delete it.

Human recipients use their experience and the context of the message to make a decision about whether or not the message is spam. They use various criteria for deciding that a message is spam such as:

- The sender's address or domain name.
- The subject of the message.
- The text of the message.

Unfortunately, in sendmail, an SMTP relay has limited access to the information in a message without significantly modifying the sendmail source code itself. During the reception of an SMTP message sendmail has access to:

- The sender's envelope return address passed in the MAIL From: command.
- The recipient's envelope address passed in the RCPT To: command.
- The host name the SMTP client announces in the HELO or EHLO command.
- The client's IP address and port number from the socket structure.

Information about the client derived from the above information and DNS.

Once the message is accepted for delivery, sendmail has access to additional information, but it is of either limited use, or impossible to use without modifying the sendmail source code. Because of this, using sendmail to filter spam is best limited to the information available during the reception of an SMTP message.

A major problem with rejecting spam using arbitrary filtering criteria is that desirable messages may be rejected along with the unwanted spam. This may range from an inconvenience for a user, to the loss of business due to rejecting a potential sales lead or business opportunity, to an extreme of a legal lawsuit charging violation of the spammer's constitutional free speech rights.

Criteria For Accepting Mail

The criteria used to accept local sender addresses with the rulesets presented in this paper are:

- Local host names
- Local domain name or local virtual domain names

Mail being sent either to or from this host or domain should be accepted. For a standalone mail host this is simple: Does the sender or recipient address contain my host name? A corporate SMTP relay is also simple: Does the sender or the recipient address's host names end with our domain? For a company with multiple domains or an ISP the test is slightly more complex because there are multiple domains that should be accepted as local. You must also make sure the local addresses are truly local and not non-local local addresses such as:

```
user%spam.dom@my.dom
```

Criteria For Rejecting Mail

The criteria that can be used to reject spam mail are:

- Host and domain names of known spam sites
- Bogus sender or host domain
- Bogus sender address
- Bogus user address
- IP networks and addresses of known spam hosts
- Invalid DNS host name
- Spam addresses hidden behind a local name

Some of these are good, but others run the risk of bouncing valid messages.

Known Spam Sites

The most straight-forward criterion for rejecting mail is to reject mail from known spam sites. This is simple to do. Make a list of user addresses, host names or domain names that you deem to be spam only sites or addresses. When the sender address or sending host is one of these spam addresses, it is rejected. While straight-forward, this is the most easily and frequently circumvented filter. Many spam sites do not give valid sender addresses to avoid getting bounce messages to bad recipient addresses, and to avoid getting spam or flame mail in return.

Bogus Sender or Host Domain

The next obvious criterion to filter on is bogus sender, host or domain name. When an invalid sender address is given, the host or domain name are frequently not in DNS. Rejecting mail from hosts and domains that can not be looked up in DNS will get rid of this class of mail. But simply looking up a host or domain name in DNS may cause a site to reject valid mail since many sites have incomplete DNS information. If a sender's host name is not registered in DNS, this criterion will cause mail from this sender to be bounced. A site can choose that this is an acceptable rejection.

A more careful approach is to search the host name and check if the sub-domain or parent domain is valid. If a domain match is found, the host name should be treated as a valid host name.

Bogus User Address

A desirable criterion for rejecting mail is to filter on bogus user address. However, testing for a bad user address is much harder because, short of sending a message to that user address, there is no reliable way to check the validity of the address. A simplistic test for a bad user address might be to connect to the sender's SMTP server and use either the SMTP VRFY or RCPT command to check the address. If the server does local delivery of the message then this would work well.

However, the reality is that most of the SMTP servers are simply SMTP relays that forward the message to an internal host. These hosts will usually accept mail for any address that ends with the local domain(s).

Next you might test the next host to which the message would be relayed. Unfortunately, most SMTP relays either do not return information about to which host the mail will be forwarded, or do not allow connections to that next host.

The other issue with using this criterion is that sendmail does not have any built-in feature that would allow the sender's complete address to be tested. Rejecting bad host or domain names only is an accepted limitation of this paper.

IP Networks and Addresses of Known Spam Hosts

The next set of criteria are based on information about the SMTP client that is connecting to the SMTP server. When a client connects to sendmail, sendmail receives information about the client:

- The host name given in the SMTP HELO or EHLO command
- The IP address of the client from the socket structure
- The TCP port number of the client from the socket structure

The first test is to check the host name given in the SMTP HELO or EHLO command as well as the DNS name of the SMTP client against the list of known spam sites. This is done by running the client host name against the same set of tests that you used for the sender's envelope address.

The second test is to check the IP address to see if it is a known spammer IP address or network. To do this, a list of known spammer IP address and networks

is created. If the IP address matches one of these IP addresses or is a host on one of the known spammer IP network address ranges then the message is rejected.

Rejecting on specific IP addresses is reasonably safe. If a spam site is sending mail from their host, all mail from that host is probably spam. Rejecting mail based on an IP network is more risky because the spammer may be using an IP address allocated to an independent ISP. If you reject mail based on the ISP's IP network number, then you will not only reject mail from the spammer, but also all of the other valid users of that ISP.

For example, if you rejected mail from aol.com's or netcom.com's networks you would block a lot of spam mail, but you would also block a lot more valid email.

Three additional tests are based on the PTR records for the client's IP address. First do a reverse lookup of the client's IP address to check for a valid PTR record. If no PTR record is found then reject it as an invalid or unauthenticated IP address. Next test if the PTR record matches the client host name given on the SMTP HELO or EHLO command. Third, do a double reverse lookup of the IP address to check that the IP address used is actually the IP address for the host name returned by the PTR record.

All of three of these tests risk bouncing more valid messages than spam messages. For various reasons a significant percentage of SMTP clients either do not have PTR records or the PTR record does not agree with the client SMTP host name or with the A records for that host. This could be because of poor DNS administration or because SMTP clients announce themselves as the domain.

However, where these tests might make sense is for messages from specific hosts or domains where you either control the domain, or where your relationship with the domain administrator would permit incorrect DNS information to be corrected. For example, if acme.com and abc.com have a business relationship, it might make sense to check that mail with a sender address of acme.com actually has consistent DNS information. If the SMTP client host name or IP address do not agree for these networks then reject the message. This test is more applicable to rejecting forged email, not rejecting spam.

Invalid DNS Host Name

Test for invalid host names by canonicalizing the address. A trailing dot is added to valid hostnames when sendmail 8 looks it up in DNS with $[hostname$] in the rulesets. Testing for this trailing dot will reject invalid names. Unfortunately, not all hostnames are registered in DNS. By treating the domain portion of the hostname as a search path, the host's sub-domain, or parent domain can be used to validate the hostname.

Spam Addresses Hidden Behind a Local Name

One technique spammers use is to hide their domain behind a local domain:

```
user%spam.dom@abc.com
```

These addresses can be rejected by borrowing the rules in ruleset SO. These rules check that an address truly is local. Extend them to test for local domains as well as domains the relay treats as local, such as the masquerade domains.

Protecting an SMTP Relay

Many sites that have implemented and paid for robust SMTP relays with high bandwidth Internet connections are being used as spam relays. A spam site will open a single connection to the relay and send a single copy of the message with a list of hundreds or thousands of recipients. The SMTP relay then delivers the messages to the individual recipients.

This causes technical problems including congestion of the SMTP relay queues, impacting the CPU performance and throughput of the SMTP relay, saturation of the Internet link, and generation of excessive undeliverable bounce messages. It causes staff problems, including time spent by postmasters and postmistresses dealing with the bounce messages, and the complaints sent to the owners of the SMTP relay from the spam's recipients. It damages the company's reputation and causes loss of good will when outside people mistakenly assume that the company encourages spam.

Criteria For Refusing To Relay SMTP Mail

Spam has many criteria that sendmail can use to reject relay spam mail:

- Is the sender or recipient a user in the local domain?
- If the sender is a user in the domain, is the client a connection from an IP address inside the domain?
- Are the recipients truly local addresses?
- Are the recipient addresses valid addresses?

Some of these criteria are good; others must be rechecked to make sure they are valid messages.

The major criteria of whether a message is valid, and should be relayed, or is spam mail that should be rejected, is if the sender and/or the recipient are part of the SMTP relay's domain(s).

The simplest case is to check that the recipient address is a local recipient. Mail going to a user within the domain should be accepted because they are part of the domain. The same criteria for accepting a sender as a local address are applied to the recipient address.

Checking the sender address as a local address to decide if the message should be relayed must be done carefully. If the sender address is a user inside the domain, and the message is being relayed either to the

outside world, or to another user within the domain, then the message should be forwarded. You should not blindly trust the sender's address because the spammer can simply forge a local sender address. Further checking is called for.

See if the client's IP address originates within the domain. If the client's IP address is not part of the domain, you can assume that the sender's address has been forged. (Restricting the sender address on the basis of the sender's IP address also has the side benefit of reducing or eliminating forged email messages from outside the domain.) Where this may not be true is with traveling or remote users. These users may send mail from outside the domain to the STMP relay.

There are two ways to deal with this, the permissive case and the restrictive case. The permissive case is where blocking on a range of IP addresses or IP networks might make sense. For example, if the domain has an exclusive or preferred ISP and does not use others then large ISPs such as compuserv.com, aol.com and netcom.com could be blocked. If the domain had a policy that employees would not send business email from aol.com, then any mail with a sender mail address with the local domain and a client IP address originating from an aol.com domain could be safely rejected. This would not block all spam mail with a local sender address coming from the outside, but it could block the most flagrant spam relay problems.

A more restrictive approach is to only allow specific ISP networks to send mail with a sender address with the local domain. Acme.com only uses the gadget.net ISP. If the client IP address was part of gadget.net's IP network then the message would be accepted. All other local sender addresses with outside IP address would be rejected. This is the approach taken in this paper.

Restricting Senders To Specific Recipient Addresses

A final issue in dealing with spam is spammers sending to internal aliases and mailing lists. A related activity is mail bombing where a user is sent a large number of messages slowing down the SMTP relay and filling up the spool directory on the mail host. In both of these cases the solution is to use both the sender and the recipient address in deciding to accept or reject the message. If the rulesets for restricting relaying is general enough so that you can test the relationship between the sender and the recipient addresses, then these same rulesets can be used with the protected addresses as special cases of relay addresses.

It is simple to reject spam to internal aliases and mailing lists by defining a list of internal addresses which are strictly internal. If mail for one of these addresses comes from outside the domain, it should be rejected.

There are several ways to reject mail-bomb mail:

- Treat the mail-bomber as a spammer and reject all mail from that sender.
- Define a class of protected users and reject mail from selected senders to those protected users. This is the approach that Eric Allman has taken.
- Selectively match specific recipients with specific senders. This is the approach I take in this paper.

Implementation Philosophy:

The initial set of check_* rulesets I wrote were based on the simple examples given by Eric Allman. As I explained these rulesets, and their uses in class, student's questions motivated me to add additional tests.

The first improvement made was to expand the use of databases in testing the address. By using format conventions used in other sendmail databases, such as denoting user addresses with an @ sign, and using domain names and dotted IP addresses in the key, a single database can be used for a wide variety of tests. This simplifies administration.

Other techniques were borrowed from other parts of the sendmail.cf file:

- Using a lookup focus for database queries.
- Marking address to separate or identify addresses.
- Recursively calling a ruleset to test all the domains in a hostname.

Integration with the rest of the sendmail.cf file was a consideration with a special focus on filtering on a relay. This includes testing for the local domain name(s) as well as other domains the relay may handle, such as masquerade domains and virtual domains.

The last design goal was to implement these rulesets as M4 templates so that they could integrate with the M4 sendmail.cf file generation philosophy. These rulesets are implemented as a series of FEATURES for simple inclusion. User configurable parameters such as database type and location are set with define('confM4_MACRO', 'value') statements. Also specific rules are included based on whether or not a specific sendmail FEATURE or M4 macro has been set.

Databases Used By check_* Rulesets

Two databases are used by the check_* rulesets presented in this paper:

- check_mail Sender addresses to be rejected
- check_fwd Sender $| recipient pairs to be accepted or rejected

Both of these databases are simple key:value pairs stored in a UNIX ndbm or Berkeley db hashed table database.

Format Of The check_mail Database

The check_mail database allows the key to be rejected to be a user address, host or domain name, or IP network or address. The value returned as the text of the message in the SMTP error message.

The format of the check_mail database is:

- The lookup key (the address)
- The value to return

Lookup Key

The key can be one of the following:

- A specific user address like `user@host.dom`. Only this address is rejected, all other addresses from host.dom are allowed. A specific user address is any key that has an @ sign in it.
- A host or a domain name like `host01.spam.dom` or `spam.dom`. All addresses with this host or domain name following the @ sign are rejected. The mail is also rejected if this is in the MAIL From: address, or it if is in the hostname of the connecting SMTP client.
- A name like `.domain_name.x`. All hostnames that end with this domain name are rejected. This rejects all mail below a domain, but not the domain itself.
- An IP network number, either one, two or three octets followed by trailing zeros, like `123.0.0.0`, or `123.123.0.0`, or `123.123.123.0`. All SMTP clients whose IP address starts with these IP network numbers will be rejected. Note that there is no check for correct class of the network entry so an entry 192.0.0.0 would reject all class C networks that start with 192.
- A specific IP network address like `123.123.123.123`. The specific SMTP client whose IP address is 123.123.123.123 .

Value Returned

The value returned can be:

- The single word OK which will accept the address and stop the ruleset.
- The single word REJECT which will return a generic SMTP error message.
- A specific message for this address which will be returned as the text of the SMTP error message (This allows you to tailor your insults to specific spam sites.)

Database Example

Table 1 shows a database example.

Updating The Database

The default name of the database is: /etc/check_mail . The source file for the database can have this name since the makemap or makedbm will append the .db or .dir and .pag extensions on to this file name. The database needs to be rebuilt each time a change is made to the sourcefile.

Use the makemap command to rebuild a hashed db database. The makemap command is a standalone command to rebuild databases provided as source code in the sendmail 8 release:

```
# makemap hash /etc/check_mail \
    < /etc/check_mail
```

To rebuild a hashed ndbm database use:

```
# makemap dbm /etc/check_mail \
    < /etc/check_mail
```

or

```
# makedbm -l /etc/check_mail \
  /etc/check_mail
```

Using the -l flag on makedbm converts all of the keys to lower case.

Format Of The check_fwd Database

The *check_fwd* database is used to test if the sender/recipient pair should be allowed or rejected for SMTP relaying.

The database key is a sender and a recipient address separated by the special token '`$|`.' If the sender and the recipient match, then the address can be accepted or rejected. The sender and recipient address can be any of the formats allowed in the check_mail database except an IP address. The sender and recipient address can also be the special key *ALL* which will match all addresses with the other half of the key.

Like the check_mail database, the value returned can either be an error message to be returned or the special key REJECT which will return a generic error message. It can be the special key OK. If the key

Key	Function
user@host.dom	Access denied for user@host.dom
host.spam.dom	Access denied for host.spam.dom
spam.dom	Access denied for dom spam.dom
123.0.0.0	Access denied for 123.0.0.0
123.123.0.0	Access denied for 123.123.0.0
123.123.123.0	Access denied for 123.123.123.0
123.123.123.123	Access denied for 123.123.123.123

Table 1: Database Example.

returned is OK, then the sender and recipient are accepted without applying additional tests.

Database Examples:

The lookup key:

```
user1@host.dom $| user2@x.abc.com
```

would block user1@host.dom from sending to the specific user2@x.abc.com The lookup key:

```
user@host.dom $| x.abc.com
```

would block user@host.dom from sending to any user at x.abc.com The lookup keys:

```
user@host.dom $| .abc.com
user@host.dom $| abc.com
```

would block user@host.dom from sending to any user in the abc.com domain. The first key blocks mail to all hosts in the domain; the second blocks mail to the domain itself. The lookup key:

```
host.dom $| user@abc.com
```

would block any user at host.dom from sending to user@abc.com. The lookup keys:

```
.abc.com $| alias@abc.com  OK
abc.com $| alias@abc.com   OK
*ALL* $| alias@abc.com     REJECT
```

would limit senders who can send to an internal alias to users in the abc.com domain.

The first two keys allow mail from any hosts in the abc.com domain or the domain itself to the alias. The third key blocks all mail from all other domains to the alias with the generic SMTP error message ‘‘Access denied.’’

Benefits Of Using Databases

The benefit of the lookup keys flexibility is that a single database can be used in several places. If the key is a user address, it can be applied to a sender or recipient. If the key is a host or domain, it can be applied to the sender or recipient, but it can also be applied to the client hostname or SMTP HELO or EHLO name. If the key is an IP address or network, then it is applied to the client’s IP address.

Another benefit of using a database to look up the addresses is that, as changes are made to the database, the updates are used immediately by the sendmail daemon.

Overview of check_* Rulesets

The new set of four check_* rulesets[1] allow sendmail to reject mail messages before they are delivered. The check_relay, check_mail, and check_rcpt rulesets are applied by the SMTP server during the SMTP protocol exchange with the SMTP client. The check_compat ruleset is applied after sendmail has accepted the message, but before the message is passed to a delivery agent. The results of these rulesets are used for an accept/reject decision. They are not used to rewrite the address by sendmail.

These rulesets can do anything they want. If they return a call to the $#error mailer, the message or SMTP command is rejected with the message that follows the $: part of the result, otherwise their result is ignored.

The rulesets to reject mail during the SMTP reception of a message are an important new feature because they allow mail to be rejected before it is transferred via SMTP to the destination or a relay. This off-loads the processing of the message and the generation of a bounce message by the mail host or SMTP relay.

The check_mail ruleset can be used to reject or accept the SMTP MAIL command based on the addresses given in the SMTP command as well as information about the SMTP client.

The check_rcpt ruleset is used to reject or accept a SMTP RCPT command based on the addresses given in the SMTP command. It can be used to reject mail for a specific recipient, host or domain. Since the

[1]For a short introduction on sendmail and the sendmail.cf configuration file, and rules and rulesets visit: http://www.harker.com/sendmail/overview.sendmail.html . For a short introduction on using M4 with sendmail to generate the config file visit: http://www.harker.com/sendmail/overview.M4.html .

```
LOCAL_RULESETS

# Start the named ruleset check_mail
Scheck_mail

# Put everything through ruleset S3 to focus
#   and canonicalize addresses
R$*                     $: $>3 $1

# Strip source route and pass back to ruleset
S32
R<@$+>:$+       $: $>3 $2
```

Listing 1: Defining a check_mail ruleset.

sender's address, as well as information about the SMTP client has already been determined, it can also be used to reject spam relay mail.

The check_relay ruleset is used to reject or accept a SMTP session. It is applied before sendmail accepts the TCP/IP SMTP connection. It is passed the DNS hostname for the client's IP address and the client's IP address itself separated by a '$|' (shown here folded for display purposes):

```
client.DNS.host.name $|
        client.host.address
```

If the $#error mailer is called, sendmail returns a '550 Access denied error' message for all subsequent SMTP commands.

The check_compat ruleset is different that the previous three rulesets because it is applied after the message has been accepted by sendmail, but before sendmail tries to deliver the message by calling the delivery agent. Thus, it can be applied to mail received from all sources, not just SMTP. This includes UUCP, user agents, and SMTP front-ends such as smap and smtpd. It is passed the processed sender address and the processed recipient address separated by a '$|' (shown here folded for display purposes):

```
sender@sendhost.dom $|
          recipient@rcpthost.dom
```

If the $#error mailer is called, sendmail will bounce the message as undeliverable. The drawback of this ruleset is that the bounce message is generated by the local sendmail process, not on the remote client trying to submit a message to this host.

Useful Macros in the check_* Rulesets

Sendmail 8.8 defines several macros which are useful for rejecting SMTP mail:[2]

- $&{client_name} Name of the SMTP client
- $&{client_addr} IP address of the SMTP client
- $&{client_port} Port number of the SMTP client

In these macros the SMTP client is the host trying to forward a message by connecting to this SMTP server.

Two additional existing macros are useful in the check_* rulesets:

- $&s: The hostname passed in the HELO or EHLO command.
- $&f: The resolved sender envelope address passed in the MAIL command.

[2]Use the deferred evaluation form, $&m or $&{macro_name}, to avoid having these expanded in the rule when sendmail reads the configuration file.

Implementation of check_mail Ruleset to Reject Incoming Spam

Here is the structure of check_mail ruleset:

- Focus addresses by passing to S3.
- Check non-local local with Strip_Local ruleset.
- Check local host and domain information.
- Check for valid DNS information.
- Check sender address for spam domain.
- Check macros for spam domain.

Focus Addresses by Passing To S3

The raw address is passed to the check rulesets; the addresses are unformatted and have no focus. Passing the address through ruleset S3 first put the address into the sendmail standard format:

```
user <@domain>
```

or for RFC 822 source route address:

```
<@domain> : balance of source route
```

This is called the 'focused address' with the next host in line for delivery inside the focus angle brackets, <>. In this configuration it is assumed we are not interested in the source route address, only in the final destination, so RFC 822 source route addresses are stripped.

Listing 1 shows how to define the check_mail ruleset with these rules in a sendmail.cf file generated with M4.

The M4 macro LOCAL_RULESETS includes the text that follows it in the sendmail.cf file before mailer definitions and is used for all ruleset additions in this paper.

Testing for local host or domain information:

A simple performance benefit is to accept addresses with local host or domain information. This avoids any additional processing of these addresses and eliminates their database lookups.

This local information is stored in the sendmail macros and classes:

- $w: My hostname
- $=w: Other host I accept as local
- $m: My domain name
- $=M: Domains I masquerade

The matching of these macros and classes is done with a repeated series of Domain Match Blocks (my terminology)

Domain Match Block

Listing 2 shows the rules that perform domain matching. On the Left Hand Side (LHS) of the rule, the match.pattern inside the focus of the address can

```
# Does it end with our domain, if so OK
R$+<@ match.pattern . >        $@ OK
R$+<@ $+ . match.pattern . >   $@ OK
```

Listing 2: Domain matching rules.

be a macro, $m, class, $=w, or text, abc.com. The first rule tests for the match.pattern itself. The second rule tests for all hosts and sub-domains below the match.pattern .

If the address matches the LHS pattern, then the RHS exits the ruleset and returns the symbolic name OK. The OK returned by these rules are strictly for human consumption. A ruleset returning OK shows that the address was explicitly accepted by a rule. If the address itself is returned by the ruleset, then the address fell out the bottom of the ruleset. In both cases since the $#error mailer is not returned by the ruleset, the result is discarded by sendmail and the address is accepted.

Rejecting Addresses Hidden Behind Our Domain

We want to avoid addresses hidden behind our hostname or domain name:

```
spammer%spam.dom@my.dom
```

Since this check is something we will want to do in several of the check_* rulesets, we create a custom StripLocal ruleset to do it. After calling ruleset S3, we pass all address to the StripLocal ruleset:[3]

```
# Check that the address is
# local, not user%host@my.dom
R$*<@$+> $:$>StripLocal $1<@$2>
```

The StripLocal ruleset tests for domains I do masquerading for, $=M, as well as domains looked up in the genericfrom database, $=G, and other hostnames I will treat as local, $=w.

Standard Block

Listing 3 shows the standard block of rules. The first rule tests if a single token is followed by the domain class:

```
< @ $=M . >
```

then the token passed to the dequote database, in case the sender passed a quoted string. The dequote database is a pseudo database built into sendmail which removes quotes from a string and parses the results into tokens.

The second rule tests if the focus is preceded by a non-local separator, $=O[4] and if the address ends with the domain class. If this is true, the address is not truly local so the domain is stripped and the user portion of the address is passed back to the $>StripLocal ruleset.

The third and fourth rules do the same thing except that the domain class is tested with preceding host or sub-domain names:

[3]About `$>StripLocal $>3` RHS syntax: When two rulesets are specified on the RHS, the first is called, and then the second is immediately called before executing the first rule of the first ruleset. This means that the second ruleset is processed first and the output of the second ruleset is used as the input to the first ruleset.

[4]The operators '!', '%', and '@' are the token separators contained in class O.

```
# Does it end with another domain we are known as?
#   make sure it is local
R$- < @ $=M . >              $: $(dequote $1 $) < @ $2 . >
R$* $=O $* < @ $=M . >       $@ $>StripLocal $>3 $1 $2 $3
R$- < @ $+. $=M . >          $: $(dequote $1 $) < @ $2 $3. >
R$* $=O $* < @ $+. $=M . >   $@ $>StripLocal $>3 $1 $2 $3
```

Listing 3: Standard block of rules.

```
LOCAL_CONFIG
Kcheck_mail type -o -a.REJECT /etc/check_mail
```

Listing 4: Defining check_mail in sendmail.cf

```
# Lookup user@host.dom
R $+<@$+.>  $: $(check_mail $1@$2 $: $1<@$2.>$)
```

Listing 5: Looking up entire address in check_mail database.

```
$:              Apply the rule once, do not recurse
$(check_mail    Start of the lookup in the check_mail database
$1@$2           Key used in the lookup
$: $1<@$2.>     Default rewrite if lookup fails
$)              End the lookup
```

Listing 6: The RHS syntax.

```
< @ $+. $=m . >
```

The StripLocal ruleset does not reject mail, it only removes un-needed local information. The StripLocal ruleset is applied before the preceding local checks.

Rejecting Addressees in the check_mail Database

With the address in a stripped focused format, it can be checked against the check_mail database in order to test if it should be rejected.

Before the database can be used, it must be defined in the sendmail.cf file: Listing 4 would define the check_mail database in the beginning of a sendmail.cf file generated with M4. The K line defines the database check_mail to be a database type (hash or dbm) which is located in /etc/check_mail. The -a.REJECT flag causes sendmail to append the pseudo domain .REJECT to the value returned if the look-up is successful. If the lookup fails, the look-up key is returned unmodified unless a default rewrite is specified in the RHS of the look-up rule.

With the database defined, the rule in Listing 5 looks up the entire address in the check_mail database. Listing 6 shows the RHS syntax.

Since the keys stored in the database do not have focus, the lookup key used is the user@host.domain address without focus or trailing dots. If the database lookup is successful, then the associated value is returned with the pseudo domain .REJECT appended, which is used to reject the address. If the lookup fails, a default rewrite is used to put the focus and trailing dot back into the address.

Once the address has been looked up, the next step is to test if the address should be rejected by testing if the address now ends with .REJECT If it does, use the string that precedes the .REJECT as the error message returned by the error mailer:

```
# if address ends with .REJECT
# use value returned as the
# error message
R$+.REJECT          $#error $:$1
```

Two additional tests that are made are for the fixed values OK.REJECT and reject.REJECT.[5] A return value in the database of reject.REJECT will reject the address with a default error message. A return value of OK.REJECT will accept the address without any additional processing. The ruleset will exit with the symbolic address OK.

These three rules create a block of rules which is referred to as a Reject Block of Rules (my terminology). The Reject Block is used after every database lookup in the check_* rulesets:

```
# if name is OK.REJECT,
# exit ruleset with OK
ROK.REJECT                $@ OK

# if name is reject.REJECT
# use default error message
# (shown folded)
Rreject.REJECT
          $#error $:Access denied

# otherwise use value returned
# as the error message
R$+.REJECT          $#error $:$1
```

Checking Macros Against The check_mail Database

In addition to the address itself, several macros are available for checking against the check_mail database. In particular:

$&{client_name}	The client hostname from the socket
$&s	The hostname from the HELO or EHLO Command

The separate look-up focus technique introduced in sendmail 8 is used to test these macros.

sendmail 8 Double Focus

sendmail 8 uses a double focus technique which allows a portion of an address to be checked against a database without modifying the original address. This is used in the mailertable, genericstable, and virtusertable FEATUREs. The double focus is generated by replicating the host or domain name in the focus in an initial lookup focus

```
R$* <@$+> $*    <$2> $1 <@$2> $3
```

The first focus is a lookup focus, with just the host or domain name. The second focus is the original domain focus with an @ sign followed by the host or domain name (shown here folded):

```
<lookup focus>
     stuff <domain focus> stuff
```

The lookup focus is used in a database lookup (also folded):

```
R < $+ > $* <@$+> $*
 <$(dbname $1 $)> $2 <@$3> $4
```

The result in the lookup focus can be checked for a specific pattern. If it does not match, the host or domain name in the lookup focus can be rewritten and looked up in the database again.

If it still does not match the lookup focus can be stripped to leave the original address

```
R<$+> $* <@$+>$*     $2 <@$3> $4
```

Checking A Macro with a Lookup Focus

The current macro value is prepended in a look-up focus in the address.[6] As the macro is put into the

[5] Remember, sendmail is case-insensitive in the rulesets.

[6] Remember, the original address will not start with a focus since all source route addresses have been previously stripped.

look-up focus, it is passed to the dequote database to break the macro into separate tokens for testing. This value is then looked up in the database and the result checked with a Reject Block of Rules. Finally, if the pseudo domain .REJECT is not returned, the look-up focus is stripped leaving the original address [FN.X]; see Listing 7.

Walking a Fully Qualified Domain Name

The tests so far have tested the entire lookup key. This will reject specific hosts or a specific domain, but it will not reject all of the hosts from a rejected domain. In order to do this the technique of walking the domain name is borrowed from the mailertable FEATURE of sendmail 8. A Fully Qualified Domain Name (FQDN) is passed in a lookup focus to a separate ruleset that recursively calls itself stripping the first token of the FQDN until a match is found or until only the final token remains of the FQDN.

The CheckMailDB ruleset is used to check the host part of the address. Both a specific hostname and hosts, and sub-domains below a domain can be rejected by walking the hostname. This ruleset checks the host/domain name in the lookup focus. A Reject Block of Rules is used to check the result. If a match is not found, then the first token is stripped and the domain preceded by a dot is looked up. Again, a Reject Block of Rules is used to check the result, and if a match is still not found, the remaining domain name without the preceding dot is passed back to the top of the same CheckMailDB ruleset. This cycle of stripping the first label of a domain name and calling the same ruleset again continues until only a single token remains. If the hostname makes it to the end of this process, then neither the hostname or any of its parent's domains have been blocked, and the recursive call to the ruleset unwinds as each ruleset call returns to the previous ruleset call; see Listing 8.

The flow of the successive calls to the CheckMailDB ruleset is as follows:

```
1st call CheckMailDB input:
        <host.sub.dom> org_addr
  Look-up host.sub.dom
  Lookup .sub.dom
  2nd call CheckMailDB input:
             <sub.dom> org_addr
     Look-up sub.dom
    Lookup .dom
    3rd call CheckMailDB input:
                <dom> org_addr
       Look-up dom
       Only one token
    3rd call CheckMailDB returns <dom>
  2nd call CheckMailDB returns <dom>
1st call CheckMailDB returns <dom>
```

The CheckMailDB ruleset is invoked in the check_mail ruleset by putting the hostname in a lookup focus and calling the ruleset. If the address

```
# Put the macro in a look-up focus
R$*             $: <$(dequote "" $&{macro.name} $)> $1

# look-up the macro in the check_mail database
R <$+> $+       $: $(check_mail $1 $: <$1> $2 $)

{A Reject Block of Rules}

# strip the lookup focus if .REJECT was not found
R< $* > $*      $: $2
```

Listing 7: Look-up focus followed by revert to original address.

```
SCheckMailDB
# first lookup the name as a hostname
R <$+> $+               $: $(check_mail $1 $: <$1> $2 $)

{A Reject Block of Rules}

# does the lookup name still have two or more tokens
#   separated by a dot?
# Strip the first token and lookup .domain
R< $- . $+ > $+         $: $(check_mail .$2 $: <.$2> $3 $)

{A Reject Block of Rules}

# If name start with a dot, strip the first dot and pass
#   remainder of the domain back to CheckMailDB ruleset
R< . $+ > $+            $@ $>CheckMailDB <$1> $2
```

Listing 8: Unwind domain name looking for rejects.

still has a lookup focus when it is returned by the CheckMailDB ruleset then it did not find a match in the database and the lookup focus can be stripped; see Listing 9.

In addition to walking the hostname in the address, the SMTP client hostname, $&{client_name}, and HELO or EHLO name, $&s, can also be walked by passing them to the CheckMailDB ruleset.

Rejecting Bad DNS Names

A lot of spammers use invalid sender addresses to avoid error messages and flames being sent back to them. A simple test to reject this type of mail is to check that the hostname or domain in the address is valid. This can be done by passing the address through ruleset S3 and then checking for a trailing dot. Ruleset S3 adds a trailing dot to the end of all hostnames that are found in DNS. The test can be made with the exclusive class match against class dot, $~., which contains a dot by itself; see Listing 10.

This simple test may be too heavy-handed; not all hostnames are registered in DNS. Many sites limit what internal hostnames are listed in their DNS databases or have poor administration of their DNS databases. A better test would be to walk the domain looking to see if the hostname, its sub-domain, or parent domain are valid hostnames. Since the lookup is

```
# Check the hostname and domain name against the
#   check_mail database putting the hostname
#   without the trailing dot in a lookup focus
R$* <@ $+. >            $: $>CheckMailDB <$2> $1 <@$2.>

# lookup focus no longer needed, strip it off
R< $* > $*              $: $2
```

Listing 9: Checking host and domain names against database.

```
# if the hostname in the focus does not end with a
#   dot, reject it.
R$* <@ $+ $~. > $#error $:$2.$3 unknown to DNS
```

Listing 10: Checking against the 'dot' class.

```
SWalkDNS
# Query DNS for domain in the lookup focus
R< $- . $+ > $*         $: < $[ $1 . $2 $] > $3

# Was the domain in the lookup found in DNS and
#   now ends with a dot?  If so, exit the ruleset
R< $+ . > $*            $@ < $1 . > $2

# Otherwise strip 1st token, and pass back to WalkDNS
R< $- . $+ > $*         $@ $>WalkDNS <$2> $3
```

Figure 11: Query DNS for valid hostnames.

```
# Pass non-canonicalized addresses to ruleset WalkDNS
# We strip the first token since S3 ( S96 really)
#   has already looked up the name.
R$* <@ $+ . $~. >       $: $>WalkDNS <$3> $1 <@$2 . $3>

# Is the hostname not part of a valid domain name
# i.e. does the name in the lookup focus still not
#   end with a trailing dot
R<$* $~.> $+ <@$+>      $#error $: Access denied, $4 unknown to DNS

# The hostname was part of a valid domain name
# Strip off the Qualified Domain Name in lookup focus
# adding trailing dot to hostname so all hostnames end
#   with a dot
R< $* > $+ <@ $+>       $: $2 <@$3.>
```

Figure 12: Calling WalkDNS and checking result.

in the DNS database, not the check_mail database, a separate WalkDNS ruleset is defined; see Listing 11.

The rules that call the WalkDNS ruleset and check the result are shown in Listing 12.

Rejecting Mail from Specific IP Addresses

IP addresses are used as keys in the check_mail database. These keys can be an IP network number:

```
123.0.0.0
123.123.0.0
123.123.123.0
```

or a specific IP network address:

```
123.123.123.123
```

The client's IP Address is taken from the socket and is stored in ${client_addr}; see Listing 13.

Implementation Of check_rcpt Ruleset To Reject Relay Mail

The structure of the check_rcpt ruleset follows the structure of the check_mail ruleset. Much of the checking for local user and host information is the same, with the exception that the recipient address passed in the SMTP RCPT command is being tested, rather than the sender address. If the recipient ends with one of the host or domains for which we are a relay then the address is accepted. The check_rcpt ruleset differs from the check_mail ruleset when the address is not local. It accepts the address if the SMTP client's IP address is within the local domain. The other main difference is that the sender address from the SMTP mail command is prepended to the address, separated by a $|, and the sender $| recipient pair is checked against the check_fwd database for accepting/rejecting. Structure of *check_rcpt* ruleset:

- Focus addresses by passing to S3
- Check non-local local with Strip_Local ruleset
- Check *sender* **$|** *recipient* pair for accepting/rejecting
- Check local host and domain information
- Check for valid DNS information
- Check for local client IP addresses

Checking For Local Client IP Addresses

To check if the SMTP client is a host with a local IP address, the client's IP address is put into a lookup focus and compared to local IP network numbers and addresses. Since the test is for accepting a non-local recipient from a client and most of the matching will be for entire IP network numbers (which is typically somewhat limited) a class match can be used. The class match does not need a separate file if the domain only has a few class B or C networks. If the domain needs to accept mail for lots of IP addresses or network numbers a file class definition can be used.

The only drawback of using a class is that the sendmail daemon needs to be killed and restarted each time a change is made. The IP address in the class definition can either be a dotted quad, 123.123.123.123, in which case it matches the exact IP address, or it can be one, two, or three octets with out the trailing zeros, 123, 123.123, or 123.123.123, in which case it matches all IP addresses that start with the network number.

```
# Tack on client IP address in a lookup focus
# Note: no addresses should start with focus since all
#   src routes have been stripped
R$*             $: <$(dequote "" $&{client_addr} $)> $1

# Reject specific IP addresses (all four octets)
R< $-.$-.$-.$- > $*             $: < $(check_mail $1.$2.$3.$4 $) > $5

{A Reject Block of Rules}

# Reject Class C networks (first three octets)
R< $-.$-.$-.$- > $*     $: < $(check_mail $1.$2.$3.0 $: $1.$2.$3.$4 $) > $5

{A Reject Block of Rules}

# Reject Class B networks (first two octets)
R< $-.$-.$-.$- > $*     $: < $(check_mail $1.$2.0.0 $: $1.$2.$3.$4 $) > $5

{A Reject Block of Rules}

# Reject Class A networks (first octet)
R< $-.$-.$-.$- > $*     $: < $(check_mail $1.0.0.0 $: $1.$2.$3.$4 $) > $5

{A Reject Block of Rules}

# Strip off IP address in lookup focus
R< $* > $*              $: $2
```

Listing 13: IP address manipulation.

The class can be defined with:

```
# Define a class of local IP
# addresses and networks in the
# sendmail.cf file itself
C{LocalIP}192.102.231
```

or

```
# Define a class of local IP
# addresses and networks from
# a file
F{LocalIP}/etc/LocalIP
```

Once the class is defined it can be tested with:

```
# Tack on client IP address in
# a lookup focus. No addresses
# should start with focus since
# all src routes have been
# stripped
R$*   $: <$(dequote ""
         $&{client_addr} $)> $1
# (above line folded)

# Accept our IP networks
R< $={LocalIP} .$+ > $*   $@ OK
# Accept specific IP addresses
R< $={LocalIP} > $*       $@ OK

# Strip off IP address in lookup
# focus
R< $* > $*                $: $2
```

The rule

```
R< $={LocalIP} .$+ > $*   $@ OK
```

will match either the first, first.second, or first.second.third octets of the IP address using *sendmail*'s rule matching feature of starting with one token and extending the number of tokens compared until a match is found.

Checking *sender* **$|** *recipient* Pair for Accepting/Rejecting

The main part of the *check_relay* ruleset is checking the *sender* **$|** *recipient* pair to see if they should be accepted or rejected. This follows checking the check_mail database in the check_mail ruleset, but it has an added twist in that there are two addresses, a sender and a recipient, and both need to have their domains walked. For each sender address, hostname, or domain name, a special key, sender_addr $| *sender*, is used to avoid excessive database queries. This special key is checked to see if the sender has any blocking. If the sender does not have this key, then no further checking of the sender $| recipient pair is done. If the sender does have the key, then the sender and recipient addresses are checked.

The walking of both the sender and the recipient address is done in two steps. The sender's address is walked and if the complete address, or the host or domain portion of the address has blocking, then the recipient address is then walked with the blocked sender portion. The advantage of this approach is that if a sender has no blocking, then the recipient portion of the address is not checked against the database.

Walking the Sender Address

In order to walk the sender address, the macro $&f is inserted before the recipient address in a lookup focus.

```
# prepend the sender's name in
# a lookup focus
R$*              $: <$&f> $1
```

With the sender in a lookup focus, the sender and the special key ***sender*** are checked in check_fwd db to see if sender has blocking:

```
$(check_fwd $&f : *sender* ...
```

If this pair matches the value with .REJECT appended is returned. Otherwise, the sender is kept in the lookup focus so we can continue testing the sender in the lookup focus using constructs like '`$: $1 $)`' in contexts as shown in Listing 14. If the lookup was successful, the returned address is:

```
<string.REJECT> user <@rcpt.dom>
```

If the returned value is OK.REJECT, then sender explicitly is accepted

```
R<OK.REJECT> $+        $@ OK
```

If the lookup focus ends with .REJECT, the sender has blocking and we look up the sender and the recipient *check_fwd* db like this:

```
$(check_fwd $&f $| $2 @ $3 ...
```

If this pair matches, the value is returned with .REJECT appended. Otherwise, we rewrite as

```
<sender $| rcpt hostname> rcpt addr
```

using a construct somewhat like this:

```
...  $: < $&f $| $3 > $2 <@ $3 > $)
```

in a specific context as in Listing 15.

We do not reject the address immediately. Instead we check if it is still as *sender* **$|** *recipient* pair. If it is, the sender has blocking, but it is not for the specific user@rcpthost.dom address. We walk the

```
# Reject from a specific user
# We lookup the sender and the special key *sender* in
#   check_fwd db to see if sender has blocking
R<$+> $+        $: < $(check_fwd $1$|*sender* $: $1 $) > $2
```

Listing 14 : Rejecting mail from a specific user.

recipient hostname to see if it is the specific user@senderhost.dom address that has blocking; see Listing 16. When the Walk_Rcpt ruleset returns if a match was found, the value in the lookup focus will end with .REJECT, otherwise it will be the sender's address by itself. We check it for a match with a reject block of rules.

What we have done at this point is checked the complete sender address against the recipient address, hostname, and domain name. If a match is not found, the lookup focus contains the full sender addresses. The next step is to walk the sender's hostname to see if the sender's host or domain has blocking.

```
# Walk the sender's hostname
# looking for an accept/reject
# Put the recipient addr back
# into lookup focus
R<$+@ $+ > $+
        $: $>Walk_Send < $2 > $3
# (folded here)
```

Like the Walk_Rcpt ruleset, when the Walk_Send ruleset returns, if a match was found, the value in the lookup focus will end with .REJECT. We check it for a match with a reject block of rules.

If a match is not found, then the sender did not have blocking for this recipient and the lookup focus is stripped.

The Walk_Rcpt Ruleset

The Walk_Rcpt ruleset is similar to the Check-MailDB ruleset in that it recursively calls itself each time stripping the first token and dot from the recipient hostname which follows the **$|**. in the lookup focus. Listing 17 shows an example.

The Walk_Send Ruleset

The Walk_Send ruleset follows the same steps of checking the full sender address except that the address in the lookup focus is the sender's host or domain name, not a user@host.dom address.

Like checking the full sender, the first step is to check if the sender's host or domain name has blocking; see Listing 18.

If the lookup was successful, the returned address is:

```
string.REJECT <send.host.dom>
                  user<@rcpt.dom>
```

```
# If this pair matches, reject
# Otherwise, put the sender and recipient
#   in the lookup focus
R<$+.REJECT> $+<@$+>    $: $(check_fwd $&f$|$2@$3 $: <$&f$|$3> $2<@$3> $)
```

Listing 15: Checking senders and recipients after 'REJECT'.

```
# If lookup focus has a $| then sender still has blocking, walk the hostname
R<$+$|$+> $+            $: $>Walk_Rcpt <$1$|$2> $3
```

Listing 16: Walk recipient hostname to checking blocking.

```
SWalk_Rcpt

# first lookup the name as a hostname
R <$+$|$+> $+           $: $(check_fwd $1$|$2 $: <$1$|$2>$3 $)
# If result ends with .REJECT, we found a match,
#   now exit Walk_Rcpt
R $+.REJECT             $@ $1.REJECT

# If lookup focus has a $| then sender still has blocking,
#   walk the hostname
R< $+ $| $-.$+ > $+     $@ $>Walk_Rcpt <$1 $| $3> $4

# If we have do not have two or more tokens, strip recipient from LUF
R< $+ $| $+ > $+        <$1> $3
```

Listing 17: Recursively strip tokens for checking.

```
SWalk_Send

# first lookup the sender name as a hostname with *sender* special key
R<$+> $+        $: $(check_fwd $1:*sender* $:  $) <$1>$2
```

Listing 18: Checking for blocking.

Notice that the reject message is before the lookup focus. This is to keep the sender's host or domain name intact so it can be passed to Walk_Rcpt.

If the lookup fails, the sender host or domain name did not have blocking and the returned address is:

```
<send.dom> user<@rcpt.dom>
```

We strip the first token in the lookup focus and check the *.domain* name against the *check_fwd* database; see Listing 19. If this lookup fails, we strip the first dot and call the *Walk_Send* ruleset again

```
# If address starts with a
# lookup focus with 3 or more
# tokens, sender host not found,
# walk the sender hostname
R<.$+> $+
       $: $>Walk_Send < $2 > $3
```

If at any point the database query is successful in the *Walk_Send* ruleset and lookup focus is preceded by a string.REJECT then the sender host or domain name had blocking and we need to check the recipient address and the recipient's host or domain name for explicit blocking. See Listing 20.

Validating the check_* Rulesets

These ruleset can be checked with address test mode, using `sendmail -bt`. Testing the sender in the check_mail ruleset and the recipient in the check_rcpt ruleset is as normal. You call the ruleset with the address:

```
check_mail user@spammer.dom
check_rcpt user@abc.com
```

To check a macro value or the sender in the *check_rcpt* ruleset, the macro must be set first with:

```
.D{client_addr}123.123.123.123
.Dfuser@spammer.dom
```

The ruleset is then run as normal.

Modified checksendmail

The *perl* script *checksendmail* is a program to automate the exhaustive testing of a list of addresses. It is a very useful debugging tool for *sendmail*. and was originally written by Gene Kim, Rob Kolstad, & Jeff Polk. I have modified it with two new flags:

- **-cm**: Check addresses against the check_mail ruleset
- **-cr** Check addresses against the check_rcpt ruleset

The address file has also been expanded to allow the inclusion of *sendmail* debugging macro definitions starting with **.D**to allow a sequence of addresses to be tested with specific macros set to specific values before the address is passed to the ruleset. This is very useful for doing regression testing against a series of addresses.

This modified version of checksendmail is available from: http://www.harker.com/sendmail/checksendmail. html .

Effectiveness Of *check_** Rulesets

The rulesets presented are currently in use at Pacific Bell and Harker Systems. They are relatively effective. On a technical basis, they are very effective blocking all the addresses, host and domain names, and IP addresses that are defined in the databases. The anti-relaying rules are very effective blocking spammers using a site as a spam mail exploder. On a practical basis, the anti-spam rules are less than effective. For spammers who send spam from a consistent host or domain, the ruleset is works well. Unfortunately, much of the spam is sent from throw-away accounts at large ISPs with liberal trial period policies. By the time the site is aware of incoming spam from one of these accounts, the spammer has already moved on to a new ISP account.

If a spammer is targeting a lot of users at a site and the spam can be detected quickly, then the rulesets can be very effective in blocking subsequent spam from that user.

```
# If address starts with a lookup focus with 3
#   or more tokens, sender host not found,
#   walk the sender hostname
R<$-.$+> $+          $: $(check_fwd .$2:*sender* $:  $) <.$2>$3
```

Listing 19: Strip the first token in the focus and try again.

```
# Lookup sender and full recipient
R$+.REJECT <$+> $+<@$+>    $: $(check_fwd $2:$3@$4 $: <$2:$4> $3<@$4> $)
sp 0.5
# does the lookup name still have sender and recipient?
# If so strip the first token and dot and pass remainder
#   of address back to Walk_Rcpt ruleset
R< $+ : $- . $+ > $+       $@ $>Walk_Rcpt <$1:$2.$3> $4
```

Listing 20: Sender has blocking; check recipient.

Conclusion

The rulesets presented here are designed to filter out unwanted email. Unfortunately, these techniques have limited success because it is a reactive battle and the spammers can keep one step ahead in the game. Choosing what sites to block requires careful consideration, so that valid email from a problem domain is not accidently rejected. The selected blocked sites need to be documented and agreed to by management. These tools are useful, and as more sites implement anti-spam and anti-spam relay rulesets, there will be fewer sites that the spammers can abuse.

Availability

The most current version of this discussion and rulesets will be found at: http://www.harker.com/sendmail/anti-spam .

Acknowledgments:

Thanks to Eric Allman, one of the great unsung heroes of the Internet. Email is THE ''Killer Application'' on the Internet. Sendmail makes Email on the Internet work, and Eric Allman wrote *sendmail* and has been maintaining it all these years.

Author Information

Robert Harker is a Senior Consultant with Harker Systems and responsible for teaching ''Managing Internet Mail'' (formerly ''Sendmail Made Simple,'' and ''Advanced Sendmail and Electronic Mail Domains'') and consulting for a variety of clients. Specializing in Internet electronic mail, he has over 14 years of UNIX and networking experience, and has built and managed networks for technology and transportation companies such as DHL, National Semiconductor, and Motorola. Mr. Harker has over 7 years of consulting and teaching experience, including a comprehensive series of TCP/IP networking classes given through the University of California Extension Program. His electronic mail address is harker@harker.com His web page is www.harker.com Reach him via U. S. Mail at:

Robert Harker
Harker Systems
1180 Hester Ave.
San Jose, CA 95126
harker@harker.com
408-295-6239

References

Eric Allman, ''Installation and Operation Guide For Sendmail Version 8.8,'' Sendmail.ORG, 1997.

Eric Allman, ''Anti-Spam Provisions in Sendmail 8.8,'' Sendmail.ORG, 1997.

Claus Assmann, ''Using a database in the *check_** rulesets,'' Christian-Albrechts-University of Kiel, 1997.

Robert Harker, ''Managing Internet Mail: Setting Up and Trouble-shooting sendmail and DNS,'' Harker Systems, 1997.