Taking control of your Internet email using Sendmail and Mimedefang.

Matthew Schumacher
GSEC Practical Assignment Version 1.4.c Option 1
November 8, 2004

Table of Contents.

Abstract.

This document describes many of the problems businesses face when deploying and managing an email system. This includes viruses, worms, spam, relaying, and error handling. It also describes one possible solution to these problems. While the solution outlined is sufficient for some, for others it may be a starting point for a custom mail processing solution that is tailored to fit their security or business needs.

The solution outlined meets the following mail filtering requirements:

1. Performance. All filters must perform well and not cause significant delays in accepting email.
2. Virus Scanning. All messages must be checked by a virus scanner using several different MIME parsers.
3. SPAM Filtering. All messages must be checked by one or more spam filtering technologies.
4. Recipient Validation. Mail relayed though a dedicated filter must be checked for valid recipients before it is forwarded to the destination mail server.
5. Error Handling. Messages that are found to be unacceptable for whatever reason must be rejected, as opposed to being queued for later qualification.

Audience.

This document describes many common problems with mail implementations and one possible solution to resolve them. While the problems documented in this document should be understandable by any mail administrator, the solutions described, assume that the reader already has basic Unix and Perl experience. To fully take advantage of the solution, and the flexibility it affords, the reader will need to have a good understanding of RFC1341[1], RC822[2], and Perl. More information about the RFC's (Request For Comments are standards maintained by The Internet Engineering Task Force) or Perl can be found in the links and references sections at the end of this document.

The Problems with Email Today.

1. Worms and Viruses.

Worms and viruses are a huge problem with mail systems today. According to an article called "Are hackers using your PC to spew spam and steal?" [3] by Byron Acohido and Jon Swartz, End-User computers are being used to steal information, relay spam, or compromise other hosts on the Internet. The computers are becoming a big target for crackers and spammers because other attackers are willing to pay for access to the compromised systems. One way these machines are being exploited is though viruses and worms that spread though email.

In order to understand how an email system can pass one of these malicious programs to a users PC, we first need to understand how these programs are attached to an email message.

A. What is RFC1341?

RFC1341 defines MIME (Multipurpose Internet Mail Extensions), which is a standard that allows Internet emails as defined in RFC822 to contain more than just standard plain text. This is very important because it allows users to send and receive almost anything though email. Without RFC1341 there would be no standard for attaching data to an email. MIME allows for the Content-Type message header (a part of the message that describes information about the message), which defines what type of information is stored in the body field of the email. This header can define binary data, text data, html data, another RFC822 message, any combination of the above, and more. The Content-Type “multipart/mixed” tells the MIME parser in the email client that there are multiple bodies (or parts) to the message and also defines a boundary so that the parser knows where the parts start and stop inside the message. Another Content-Type is “message/rfc822” which stores another email message, or a message inside of a message. This is commonly referred to as nested mime. As you can see, there are virtually unlimited ways to construct an email message. You can have a message with an excel spreadsheet attachment that is attached as an RFC822 message in the MIME part of another message. The ability for a mail client or virus scanner to find that spreadsheet is completely dependent on it's ability to correctly parse the nested mime information in the message.

B. MIME parsers and examples on how they can be exploited.

Mime parsers included in email clients and virus scanners seem to vary on how well they can parse complex messages. This is especially true when the message isn't completely RFC1341 compliant. Tricking the mime parser in a virus scanner into ignoring some content or tricking the mail client into running malicious code is completely dependent on the parser’s ability to gracefully handle malformed messages. This really isn't any different than expecting an operating system to process invalid tcp/ip packets without crashing or revealing private information.

Because of the infinite MIME combinations and the differences between how most MIME implementations act it is advisable to implement more than one MIME parser when doing server side email scanning.

Here are a couple of examples of email viruses that took advantage of MIME parser faults:

BadTrans.B

According to CERT Advisory AV-2001.101[4], the BadTrans.B worm placed html code inside of a MIME part, which Outlook passed to Internet explorer. The html contained code that exploited an Internet Explorer MIME handling flaw documented in Microsoft Security Bulletin (MS01-020)[5]. This flaw allows a virus or worm to define MIME header information that specifies that the attachment was one of the unusual MIME types that IE handles incorrectly causing IE to launch the attachment automatically when it rendered the html part in the message.

Not only did this worm cause chaos for Outlook users, but according to a bugtraq message posted by Jari Helenius[6] it also passed right though the Network Associates Webscan product when the worm was first released. This was because the worm used a broken MIME structure, which caused NAI's MIME parser to think that the message did not have an attachment and thus not scan the message. However, the Outlook MIME parser found the attachment and passed the malicious code to Internet explorer where it exploited the MIME flaw.

W32/Aliz

According to the Symantec Virus database[7], this worm used the same Internet Explorer MIME bug that the BadTrans.B worm did to trick Outlook into running malicious code, but it used a different MIME structure to get around the AmaViS virus scanner. AMaViS uses Reformime to parse the MIME parts and place them into files on the disk. However, a bug[8] in Reformime, caused AmaViS to skip the MIME attachment if there where any whitespace characters in front of the header lines. This bug kept the MIME parts from being written to disk. When AMaViS called the virus scanner it didn't find anything to scan, and reported that the message was clean.

CR Character Vulnerability

An example of a MIME based attack that has not yet been used by a virus is the Outlook 'CR' character vulnerability [9]. This is when Outlook treats a single 'CR' (carriage return) character as the end of a header line, where RFC822 says that 'CR' followed by a 'LF' (line feed) is used to end a header line and that CR and LF characters must not appear alone in the headers.

Attackers can take advantage of this vulnerability by ending the headers portion of the email using the CR character, then place a virus in the message headers where an RFC1341 compliant MIME parser will be looking for the CR followed by a LF to end the headers before it will start looking for any MIME parts.

C. What other ways can messages sneak viruses past a virus scanner?

Another method for getting the virus passed the mail system is to link to them in an HTML attachment usually using the iframe tag[10]. This prevents the virus from moving though the email system altogether, but as soon as the client renders the HTML, the virus is downloaded from another location on the Internet. There are a couple of ways round this: Drop or convert all HTML messages to plain text; wash the HTML MIME parts though another filter or virus scanner that tries to remove links to external sites; or you can let them though the mail system and trust that whatever solution is in place to detect viruses on web pages will catch the virus.

One of the primary security principles taught in the SANS course is defense-in-depth. That concept applied to this problem states that you are better off deploying all three of the above solutions to ensure that one way or another the virus is detected, but sometimes this is not always possible because of user or business requirements. While removing HTML from email is something that will greatly reduce the risk of viruses and worms, it may hinder the mail solution to the point that it is no longer useful. A security concept that sums up the, "security verses functionality" tradeoff states that, "Security is a direct trade-off for functionality, right up until you unplug it, it's very secure, and it doesn't work." This concept is a good reminder that security in the extreme can hinder the user every bit as much as insecure systems.

D. MIME parser testing.

GFI Software USA, Inc., which offers various security products, has an excellent email security test available at http://www.gfi.com/emailsecuritytest/, which will send 17 different infected messages to a mailbox. These messages contain exploits that are in use by worms and others that have not been used. The email security test FAQ (http://www.gfi.com/emailsecuritytest/faq.htm) describes what each test does and how it has been or could be used to infect a mail client.

Another mail scanner test available at http://www.testvirus.org/, has 26 different test messages that try to find a weak spot in virus scanners.
Much like firewalls are specifically engineered to protect a user's host from malformed packets and other attacks, a mail solution should also be able to protect the user's mail client from malformed emails and attacks. In order to accomplish this it is necessary to use a MIME parser that is specifically designed to look for ways that mime can be broken.

2. Relaying Mail.

Depending on security requirements and available budget, separating the mail filtering solution from the actual mail system may be the best option. This is the recommended method as it layers security, splits the mail load and filtering load between multiple hosts, allows for firewalls and IDS (Intrusion Detection systems) systems to be deployed between the mail servers, and can even split up the email system responsibility between different two different groups, one concentrating on email security and the other on email availability.

For more information on how to setup a mail relay please see Jason D. McLellan's paper entitled, "A Secure Sendmail Based DMZ for the Corporate Email Environment"[11].

A. Problems with relaying email.

There are several problems with how email is bounced (or rejected) in many mail implementations that can cause problems when relaying email.

Some mail implementations bounce invalid messages during the SMTP session (Simple Mail Transport Protocol, the protocol mail systems use to send message to each other) and refuse the message, while others accept the message (since the recipient validation and other checks happens later in the mail process) and forward the message to the mail-processing engine where the message is found to be invalid. An error notification message is then generated and sent to the sender of the original message.

Here is a SMTP conversation with a mail system that rejects invalid recipients:

```
220 server ESMTP Sendmail 8.13.1/8.13.11; Thu, 30 Sep 2004
helo senderdomain.com
250 server Hello client [x.x.x.x], pleased to meet you
mail from: <sender@senderdomain.com>
250 2.1.0 <sender@senderdomain.com>... Sender ok
rcpt to: <nonvaliduser@rcptdomain.com>
553 5.3.0 <nonvaliduser@rcptdomain.com>... Mail sent to invalid user
```

The 5.3.0 error on the last line tells the remote mail system that the message is rejected and why. This causes the connecting mail system to generate an error message to the sender, which simply relays the reject message in form of a bounce message. This is what the bounce message would look like:

```
   ----- Transcript of session follows -----
... while talking to server.:
>>> DATA
<<< 550 5.1.1 <nonvaliduser@rcptdomain.com>... User unknown
550 5.1.1 <nonvaliduser@rcptdomain.com>... User unknown
<<< 503 5.0.0 Need RCPT (recipient)
```

It is important to note that this message is generated by the connecting mail system and it simply relays the reject message the destination server provided.

Here is a SMTP conversation with a mail system that does its recipient validation after the message is accepted*:*

```
220 server Microsoft ESMTP MAIL Service, Version: 6.0.3790.0 ready
helo senderdomain.com
250 server Hello [x.x.x.x]
mail from: <sender@senderdomain.com>
250 2.1.0 sender@senderdomain.com....Sender OK
rcpt to: <nonvaliduser@rcptdomain.com>
250 2.1.5 nonvaliduser@senderdomain.com
data
354 Start mail input; end with <CRLF>.<CRLF>
subject: test

test
.
250 2.6.0 <SERVER-WINCCjII5A00000001> Queued mail for delivery
```

In this example the message is accepted and queued for delivery even though it's not valid. This causes the mail processing engine to generate a bounce message that is sent to the original sender. Here is an example of a bounce message:

```
This is an automatically generated Delivery Status Notification.

Delivery to the following recipients failed.

       nonvaliduser@rcptdomain.com
```

This message was generated by the destination mail server and is sent to the sender listed in the original message header.

Some problems caused by not bouncing the message before accepting it include:

1. Spam messages are rarely sent from a valid email address so the bounce message gets sent to an invalid address where it is bounced again (double bounce).

2. The junk message uses bandwidth because the entire message is sent, then accepted, then sent again in the bounce message back to the sender (if even valid). Basically, a message addressed to an invalid recipient uses twice the bandwidth than a valid message. If the message was rejected in the SMTP session it would only get to the 5th line of the SMTP session before bounced.

3. Additional load on the mail system. Why accept the message and go though the task of processing it, virus scanning it, and spam filtering it when it's not even valid?

These problems are even more complex when you place a mail relay in front of the trusted mail system. Most relays have no way of validating users and in turn forward every message that passes the spam filtering and virus scanning though to the trusted mail system. Trusted email systems then either bounce the message back to the relay mail system in the case of a SMTP level rejection, or they accept the message and send a bounce email to the original sender when the mail engine processes the message. Either way, spam addressed from invalid recipients will cause double bounces and wasted bandwidth, not to mention invalid messages make it all the way to the trusted mail system which, to some degree, defeats the purpose of filtering mail in an external relay to begin with.

B. A better solution for bouncing email.

The best way to deal with rejecting email is to do it at the SMTP level so that the connecting host always generates the bounce message. Bouncing messages at the SMTP level is also good email policy since messages are filtered at the network edge and are either accepted and sent to the trusted mail system or rejected. Invalid messages simply never make it passed the relay server at the network edge. This also reduces complexity because the mail system never sends a bounce message or any other notification to a remote user, which pretty much eliminates double bounces.

Not only can this be done for invalid recipients but also for messages rejected due to spam and viruses. Spam messages would be rejected instead of accepted, scanned, and then deleted. The system would return "554 5.7.1 Rejected because the message looks like SPAM" which tells the spammers that the mail system is not going to accept the message, while at the same time telling the sender of a valid message (false positive) that their message was rejected. This solution is better than simply deleting the spam message, because senders of valid messages are notified. Deleting spam may cause a victim of a false positive to think that their message reached the recipient when in fact it didn't.

The same principle applies to viruses. Rejecting a message consumes much less bandwidth than emailing the sender (which is usually bogus) to let them know that they sent a virus. It can also help notify users of infected systems because all attempts to relay a virus infected message will be rejected regardless if the message sender is forged or not. This also avoids sending a virus notification to a spoofed sender that doesn't actually have a virus. Many find this even more annoying than spam.

A Solution.

Implementing this solution is fairly straightforward but in order for it to be really effective it will require some Perl knowledge. Mail processing in Perl allows for flexibility to do practically anything, but with that flexibility comes leaning curve and complexity.

Here is the software chosen for this solution and why:

Unix (or a clone): These tools are not available on Windows and Unix is immune to Windows viruses. Most modern Unix systems also come with advanced file systems, Logical Volume Management, and many other features, which make them the logical choice for processing email.

Sendmail: http://www.sendmail.org

Sendmail offers a very flexible configuration format that allows the mail administrator to write custom rules and query information from different sources. Sendmail also has a plugin interface called milter. The milter interface allows Sendmail to pass a message to another program (the milter), which is already running in memory. The milter can change the message or return result information to Sendmail, which Sendmail uses to accept or reject the message. Milters have 2 distinct advantages over most mail filtering solutions. First, they can be called before the message is accepted, allowing Sendmail to reject messages during the SMTP session, and second, they can be called from a unix or tcp socket. Passing the message to a running program though a socket significantly reduces filter processing overhead because the milter stays in memory waiting for connections instead being calling from a shell. Milters can also be placed on other hosts, where they can be called over the network.

Mimedefang: http://www.mimedefang.org

Mimedefang allows for mail processing in Perl, which is an easy to understand language that is well suited for filtering. Mimedefang runs as a daemon, which makes it much faster than other Perl filters that are called from a shell. Mimedefang uses the mimedefang-multiplexor, which maintains a pool of 1 or more Perl processes that are running and ready to go. Messages that are processed by one of these running Perl processes are filtered much faster because the startup overhead required to compile and execute the Perl script, then parse any configuration files, is avoided. Mimedefang also uses the Sendmail milter interface, which allows for filtering before the message is accepted.

Spamassassin: http://www.spamassassin.org

Spamassassin is a popular spam filter that combines several spam filtering technologies in a scoring system in order to determine the likelihood that the message is spam. Spamassassin is written in Perl so it is called as a Perl module from a Mimedefang filter process. This means that spamassassin is always in memory and ready to go.

Anomy Sanitizer: http://mailtools.anomy.net/

Anomy Sanitizer has its own MIME parser and is used to disable potentially dangerous html code such as iframes and javascript. It does many of the same things as mimedefang so it may seem redundant, but having multiple MIME parsers reduces the risk of a malicious email passing though the mail filter solution to the mail client. Anomy Sanitizer is also written in Perl, so it runs under a mimedefang filter process.

Clamav: http://www.clamav.net

Clamav is an open source virus scanner that runs as a daemon. It is very fast since its virus definition database is always loaded into memory while it is running. It also has its own MIME parser that can detect viruses in RFC822 messages without an external MIME parser to extract files embedded in a message.

Software Installation.

Please note that this document does not describe the installation specifics for each software component in significant detail. This document is not meant to be a HOW-TO, but rather a guide to building an effective open source mail relay that will address the problems outlined in prior sections. Answers to questions about the software used can be found in the resources listed at the end of this document.

The following code, configuration, and syntax are for building a mail relay designed to be placed in front of an existing mail solution. If a single mail host is preferred, then this solution can be adapted to run on a single server by simply adding some sort of local delivery agent and pop3/imap server. The Carnegie Mellon Cyrus server (http://asg.web.cmu.edu/cyrus/) is a good solution because it is very fast, scales well, and is stable.

Sendmail:

First compile Sendmail with ldap, milter, sasl, and ssl support:

```
tar xvzf sendmail.<version>.tar.gz
cd sendmail-<version>
```

Copy the following to <sendmail-source>/ devtools/Site/site.config.m4:

```
define(`confDEPEND_TYPE', `CC-M')
define(`confSM_OS_HEADER', `sm_os_linux')
define(`confMANROOT', `/usr/man/man')
define(`confLIBS', `-ldl')
define(`confEBINDIR', `/usr/sbin')
APPENDDEF(`confLIBSEARCH', `crypt nsl')
APPENDDEF(`confENVDEF', `-DSASL -DSTARTTLS -DMILTER')
APPENDDEF(`conf_sendmail_LIBS', `-lsasl2')
APPENDDEF(`confMAPDEF', `-DLDAPMAP')
APPENDDEF(`confLIBS', `-lldap -llber')
APPENDDEF(`confINCDIRS', `-I/usr/local/include -I/usr/include/sasl')
APPENDDEF(`confLIBDIRS', `-L/usr/local/lib -L/usr/lib/sasl2')

define(`confLD', `ld')
define(`confMTCCOPTS', `-D_REENTRANT')
define(`confMTLDOPTS', `-lpthread')
define(`confLDOPTS_SO', `-shared')
define(`confSONAME',`-soname')

ifelse(confBLDVARIANT, `DEBUG',
dnl Debug build
`
	define(`confOPTIMIZE',`-g -Wall')
',
dnl Optimized build
confBLDVARIANT, `OPTIMIZED',
`
	define(`confOPTIMIZE',`-O2')
',
dnl Purify build
confBLDVARIANT, `PURIFY',
`
	define(`confOPTIMIZE',`-g')
',
dnl default
`
	define(`confOPTIMIZE',`-O2')
')
```

Now compile and install:

```
./Build –c
./Build install
cp -r cf /etc/mail
cd libmilter
./Build install
cd ..
cp sendmail/aliases /etc/mail
mkdir /var/spool/mqueue
chown root:root /var/spool/mqueue
chmod 700 /var/spool/mqueue
mkdir /var/spool/clientmqueue
chown smmsp:smmsp /var/spool/clientmqueue
chmod 770 /var/spool/clientmqueue
```

Compile and install Spamassassin:

```
perl -MCPAN -e shell
o conf prerequisites_policy ask
install Net::DNS
install Mail::SpamAssassin
quit
```

Install Anomy Sanitizer:

```
tar xvzf anomy-sanitizer-<version>.tar.gz
cd anomy/bin
mv Anomy/ /usr/lib/perl5/<ver>/
mv * /usr/local/bin/
```

All that is needed for MIMEDefang to use the MIME parser in Anomy Sanitizer is for Anomy Sanitizer to be installed.

Add user for clamav and mimedefang:

```
useradd defang
groupadd defang
```

Compile and install clamav:

```
tar xvzf clamav-<version>.tar.gz
cd clamav-<version>
./configure –with-user=defang –with-group=defang
make
make install
```

Edit the /usr/local/etc/freshclam.conf file and remove the example line:

```
 # Comment or remove the line below.
 Example
```

Update the clamav database:

```
Freshclam
```

Edit the /usr/local/etc/clamav.conf and remove the example line. Set 'User' to 'defang', set 'LocalSocket' to '/var/spool/MIMEDefang/clamd.sock', and set 'TemporaryDirectory' to '/var/spool/MIMEDefang':

```
LocalSocket /var/spool/MIMEDefang/clamd.sock
User defang
```

Start clamd:

```
Clamd
```

Confirm that clamav is running:

```
ps -ef | grep clamd
```

Now compile and install Mimedefang:

```
perl -MCPAN -e shell
install MIME::Tools
tar xvzf mimedefang-version.tar.gz
cd mimedefang-<version>
./configure
make
make install
cp examples/init-script /etc/rc.d/rc.mimedefang
```

Mimedefang and Clamav perform much better if they extract MIME parts to a ramdisk. This is not required, but is recommended because it pretty much removes disk I/O from the mail process since all message extraction is done in memory. Make sure that the ramdisk is large enough contain all of the messages that may be processed at any one time. Most mail systems will be fine with 128MB as it will take quite a few concurrently processing messages to exceed that amount of space.

Configure Mimedefang to extract messages to a ramdisk.

Put this ramdisk statement in the /etc/fstab file: (Linux syntax)

```
none          /var/spool/MIMEDefang   tmpfs mode=700,size=128M,uid=defang,gid=mail 0 0
```

Now mount the disk:

```
mount /var/spool/MIMEDefang
```

Configure mimedefang:
Edit the /etc/mail/mimedefang-filter and find the filter_begin() function around line 131. Edit the function to reject the message instead of discarding it.

Change:

```
if ($FoundVirus) {
  md_graphdefang_log('virus', $VirusName, $RelayAddr);
  md_syslog('warning', "Discarding because of virus $VirusName");
  return action_discard();
}
```

To

```
if ($FoundVirus) {
  md_graphdefang_log('virus', $VirusName, $RelayAddr);
  md_syslog('warning', "Bouncing message because of virus $VirusName");
  action_bounce("Message contains $VirusName virus, rejected");
  return action_discard();
}
```

Now edit the filter_end() function around line 286.

Change:

```
   # Spam checks if SpamAssassin is installed
   if ($Features{"SpamAssassin"}) {
      if (-s "./INPUTMSG" < 100*1024) {
         # Only scan messages smaller than 100kB.  Larger messages
         # are extremely unlikely to be spam, and SpamAssassin is
         # dreadfully slow on very large messages.
         my($hits, $req, $names, $report) = spam_assassin_check();
         my($score);
         if ($hits < 40) {
            $score = "*" x int($hits);
         } else {
            $score = "*" x 40;
         }
         # We add a header which looks like this:
         # X-Spam-Score: 6.8 (******) NAME_OF_TEST,NAME_OF_TEST
         # The number of asterisks in parens is the integer part
         # of the spam score clamped to a maximum of 40.
         # MUA filters can easily be written to trigger on a
         # minimum number of asterisks...
         if ($hits >= $req) {
            action_change_header("X-Spam-Score", "$hits ($score) $names");
            md_graphdefang_log('spam', $hits, $RelayAddr);

            # If you find the SA report useful, add it, I guess...
            action_add_part($entity, "text/plain", "-suggest",
                        "$report\n",
                        "SpamAssassinReport.txt", "inline");
         } else {
            # Delete any existing X-Spam-Score header?
            action_delete_header("X-Spam-Score");
         }
      }
   }
```

To:

```
   # Spam checks if SpamAssassin is installed
   if ($Features{"SpamAssassin"}) {
      if (-s "./INPUTMSG" < 100*1024) {
         # Only scan messages smaller than 100kB.  Larger messages
         # are extremely unlikely to be spam, and SpamAssassin is
         # dreadfully slow on very large messages.
         my($hits, $req, $names, $report) = spam_assassin_check();
         my($score);
         if ($hits < 40) {
            $score = "*" x int($hits);
         } else {
            $score = "*" x 40;
         }
         # We add a header which looks like this:
         # X-Spam-Score: 6.8 (******) NAME_OF_TEST,NAME_OF_TEST
         # The number of asterisks in parens is the integer part
         # of the spam score clamped to a maximum of 40.
         # MUA filters can easily be written to trigger on a
         # minimum number of asterisks...

         action_change_header("X-Spam-Score", "$score");
         action_change_header("X-Spam-Report", "$hits ($req req)  $names");

         if ($hits >= $req) {
            md_graphdefang_log('spam', $hits, $RelayAddr);
            action_change_header('Subject', "[SPAM] $Subject");
         }

         # if we're quite confident it actually IS spam, just bounce it
         if ($hits > 8) {
           action_bounce("Message seems to be spam, rejected");
           return action_discard();
         }

      }
   }
```

This will cause mimedefang to reject messages that look like spam giving legitimate users some form of notification that the message was dropped while announcing to the spammers that their spam attempts are not going to be accepted.

Now start mimedefang:

```
/etc/rc.d/rc.mimedefang start
```

Configure Sendmail to use Mimedefang:

Put the following in the /etc/mail/sendmail.mc file. (Watch for line wraps):

```
divert(-1)
dnl This is the sendmail macro config file. If you make changes to this file,
dnl you will need to recreate the sendmail.cf with the command below
dnl
dnl       m4 /etc/mail/sendmail.mc > /etc/mail/sendmail.cf
dnl

include(`/etc/mail/cf/m4/cf.m4')
OSTYPE(linux)

define(`confSMTP_LOGIN_MSG', `Mail Server')

INPUT_MAIL_FILTER(`mimedefang', `S=unix:/var/spool/MIMEDefang/mimedefang.sock, F=T,
T=S:1m;R:1m')

FEATURE(`use_cw_file')
FEATURE(`access_db')
FEATURE(`blacklist_recipients')
FEATURE(`mailertable')
FEATURE(`dnsbl', `relays.ordb.org', `Rejected - see http://ordb.org/')
FEATURE(`dnsbl', `relays.visi.com', `Rejected - see http://relays.visi.com/')
FEATURE(`dnsbl', `sbl.spamhaus.org', `Rejected - see http://spamhaus.org/')
FEATURE(`dnsbl', `list.dsbl.org', `Rejected - see http://dsbl.org')
FEATURE(`dnsbl', `cbl.abuseat.org', `Rejected - see http://cbl.abuseat.org')
FEATURE(`dnsbl', `bl.spamcop.net', `Rejected - see http://spamcop.net')

O MaxRecipientsPerMessage=10
O MaxMessageSize=8000000

MAILER(local)
MAILER(smtp)
```

Create the Sendmail config file (sendmail.cf):

```
m4 /etc/mail/sendmail.mc > /etc/mail/sendmail.cf
```

Create config files needed for Sendmail:

```
touch /etc/mail/local-host-names
newaliases
touch /etc/mail/access
makemap hash /etc/mail/access < /etc/mail/access
```

Configure Sendmail to relay your domain to your internal server:

```
echo “domain.org” > /etc/mail/relay-domains
echo “domain.org   <tabs>   smtp:[<dest server ip>]” > /etc/mail/mailertable
makemap hash /etc/mail/mailertable < /etc/mail/mailertable
```

Start Sendmail:

```
sendmail -L sm-mta -bd -q1h
sendmail -L sm-msp-queue -Ac -q30m
```

Use telnet to talk to the SMTP server and pass it the eicar test virus string documented at http://www.eicar.org/anti_virus_test_file.htm:

```
$ telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 Mail ESMTP Server
helo example.org
250 linux.example.org Hello localhost [127.0.0.1], pleased to meet you
mail from: user@example.org
250 2.1.0 user@example.org... Sender ok
rcpt to: user@example.org
250 2.1.5 user@example.org... Recipient ok
data
354 Enter mail, end with "." on a line by itself
subject: test

<Place EICAR-STANDARD-ANTIVIRUS String Here>

.
554 5.7.1 Message contains Eicar-Test-Signature virus, rejected
quit
221 2.0.0 linux closing connection
Connection closed by foreign host.
```

Now use telnet to talk to the SMTP server and pass it the gtube test string documented at http://spamassassin.apache.org/gtube/:

```
$ telnet localhost 25
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
220 Mail ESMTP Server
helo example.org
250 linux Hello localhost [127.0.0.1], pleased to meet you
mail from: user@example.org
250 2.1.0 user@example.org... Sender ok
rcpt to: user@example.org
250 2.1.5 user@example.org... Recipient ok
data
354 Enter mail, end with "." on a line by itself
subject: test

<Place GTUBE-STANDARD-ANTI-UBE-TEST-EMAIL String Here>

.
554 5.7.1 Message seems to be spam, rejected
quit
221 2.0.0 linux closing connection
Connection closed by foreign host.
```

If the previous two tests resulted in a rejected message as shown in the example then virus scanning and spam filtering has been successfully installed. The next step is to filter invalid recipients.

Filtering Invalid Recipients.

Filtering invalid recipients on a relay server is difficult because it lacks the account information required to determine if the recipient address is valid or not. Checking for valid addresses requires the relay to query account formation in real time as it is processing a message.

There are a couple of common ways to go about checking on this information:

1. Some mail filters (including Mimedefang) can use SMTP to ask the destination mail server if the recipient is valid before accepting it. This method only works on servers that reject invalid recipients during the SMTP session such as Sendmail or Exchange server 2003. This method can be slow because the relay needs to connect to the destination mail system's SMTP server to check on each recipient while the sending SMTP server waits for the result.

2. If the user account information is stored in a Sendmail supported database such as LDAP then it may be possible to write a custom Sendmail rule to check this information. This method is very fast, but requires the mail relay to have access to account information on the internal mail system, which may not be acceptable. Here is some example code written by the author that causes Sendmail to check for recipients in Active Directory using the Sendmail LDAP map feature.

```
define(`confLDAP_DEFAULT_SPEC', `-h "<adserver>" -b "dc=domain,dc=org" -d
"user@domain.org" -P "/etc/mail/ad.secret" -l5')

LOCAL_CONFIG
Kemaillookup ldap -n -v objectClass -T.TMPFAIL -k (&(proxyAddresses=%0))
Kexchangedomains hash /etc/mail/exchangedomains

LOCAL_RULESETS

SLocal_check_rcpt
R$*                     $: $>3 $1
R$+ < @ $+ . >          $: $1 < @ $2 . > $(exchangedomains $2 $: $)
R$+ < $+ . > CHECK      $: $1 < $2 . >   $(emaillookup smtp : $1 $2 $: . DENY $)
R$+ . DENY              $#error $@ 5.7.1 $: "Mail sent to invalid user"
R$+ . TMPFAIL           $#TEMP $@ 4.7.1 $: "please try again later"
R$+ < $+ > $*           $: $1 < $2 >
```

3. Write a socketmap server. Sendmail 8.13.x now includes an interesting new feature that allows you to define a server as a database macro. This would allow you to use a rule very similar to the one above except it would query a custom written server instead of an LDAP directory. The socketmap protocol is trivial so it should be easy for an experienced programmer to write a socketmap server. Please see the documentation in the Sendmail operations guide included in the Sendmail source distribution.

Since the example solution is already using Mimedefang, the recipient filtering example will use the SMTP filter method.

Open the /etc/mail/mimedefang-filter file and add the following at the bottom after the filter_end() function:

```
sub filter_initialize {

  require DB_File;
  use Fcntl;
  tie %relaydomains, "DB_File", "/etc/mail/mailertable.db", O_RDONLY;

}

sub filter_recipient {

  my($recip, $sender, $ip, $host, $first, $helo, $rcpt_mailer, $rcpt_host, $rcpt_addr) = @_;

  ($user,$domain) = split /\@/, $recip;

  if( $relaydomains{$domain} =~ /^smtp:\[(.+)\]/){
    return md_check_against_smtp_server($sender, $recip, "<filter host>", $1);
  } else {
    return ("CONTINUE", "OK");
  }

}
```

Tell Mimedefang to call the filter_recipient function:

In your /etc/rc.d/rc.mimedefang file change:

```
# MX_RECIPIENT_CHECK=no
```

To

```
MX_RECIPIENT_CHECK=yes
```

Then restart Mimedefang:

```
/etc/rc.d/rc.mimedefang restart
```

As you can see the Mimedefang implementation is pretty simple. When the filter is first started it opens the mailertable database, then filter_recipient() is called on every message where it strips out the domain part of the recipient then checks to see if there is a record in the mailertable database for that domain then calls md_check_against_smtp_server() against that SMTP server and returns the result.

After this example code is in place the relay server will check to make sure that the recipient is valid before accepting a message, but only for domains defined in the mailertable database.

Spam filtering.

Spamassassin uses a point system to determine if a message is spam or not. Each message is processed by rules that have point values assigned to them. When all of the rules have been checked, Spamassassin adds up the points for each rule matched, which describes to Spamassassin how much the message looks like spam. For example if you define all messages that score greater than 5 as spam, then assign 3.5 points to a rule that looks for the word 'Viagra' then another 3.5 points to another rule that looks for a valid 'User-Agent' header, then all messages that match both rules will be rejected because the spam score adds up to 7, but messages that only match one rule will be accepted. This allows a user to email his doctor about Viagra, as long as there isn't any other compelling reason to mark the message as spam.

The more spam filtering technologies that spamassassin uses the more qualified the message becomes. Spam messages will have very high scores and ham messages (non-spam) have very low scores. This makes it possible to lower the spam score threshold without causing massive amounts of false positives (legitimate messages identified as SPAM).

Since Spamassassin is installed and Mimedefang detects it at runtime the basic Spamassassin rules are already in place. This includes the many spamassassin rules documented at http://spamassassin.apache.org/tests_3_0_x.html and the RBLS (Real Time Blacklists).

Here is a list of some of the more popular spam filtering technologies spamassassin uses and how to install and configure them.

A. Real Time Blacklists.

Real time blacklists are DNS based databases of black listed servers, that can be queried to find out if a connecting host is listed. There are many RBLs available, each with different purposes and legal requirements. For information on how RBLS work please read the documentation at http://www.dnsbl.us.sorbs.net/.

RBLs can be implemented in Sendmail or Spamassassin. The difference is that Sendmail will reject mail from all hosts listed in the defined RBLs, but Spamassassin will use the score associated with the RBL rule along with all of the other rules to determine if the message is spam. Basically, Spamassassin uses RBLs as another method to qualify spam where the sending host being listed in an RBL may not, in it's own right, cause the message to be rejected.

Using a combination of the above can be very affective if RBLs that list spam sources are defined in Sendmail which causes the message to be rejected, but RBLs that list dynamic addresses or other information are defined in Spamassassin. This works well because mail systems that are connected via DSL and cable modems are not necessarily rejected, but are more prone to sending spam from virus or worm infected machines so they are scored accordingly.

For information on how to work with RBLs in Spamassassin please see: http://wiki.apache.org/spamassassin/DnsBlocklists and for information on how to setup a RBL in Sendmail see: http://www.sendmail.org/m4/features.html.

For a list of available RBLs please see: http://openrbl.org/zones.htm

B. Bayesian Filters.

Bayesian filters work on statistics gathered from known spam or ham messages. Messages are analyzed to determine how much the message resembles the spam and ham messages the filter learned from.

Bayesian filters work well, but they must have an idea of what spam and ham messages look like before they can be effective. Spamassassin includes a tool called sa-learn that is used to teach Spamassassin about spam and ham messages. Here is an example of how to use it:

```
su defang -c "sa-learn --spam --mbox --showdots samplespam.mbox"
su defang -c "sa-learn –ham --mbox --showdots sampleham.mbox"
```

Please notice that in this example the input files are in the standard Unix mbox[12] format (basically rfc822 messages concatenated together in a plain text file) and the command is run as the defang user since the databases sa-learn creates need to be owned by the same user Spamassassin runs as.

Spamassassin can also auto-learn from messages it processes. Please see the Spamassassin documentation for instructions on how to enable this feature. http://spamassassin.apache.org/full/3.0.x/dist/doc/Mail_SpamAssassin_Conf.html

For more information about Bayesian filters please read “What You Need to Know About Bayesian Spam Filtering”[13] by Heinz Tschabitscher.

C. Razor

Razor is a message fingerprinting system written by Paul Vipul. According to his documentation[14], Razor “works by computing signatures on the body of the content and checking these signatures against a database of known spam.”

Razor does not send your email to the razor database, it sends a signature generated from your email to the database to see if it matches a signature of known spam. If the message signature is found to match a known spam message then the points associated with the razor rule in Spamassassin is added to the messages total spam score.

Since Spamassassin detects razor at runtime all that is needed to use it is install it. The sources are available at: http://razor.sourceforge.net/.

Installing razor:

```
tar xvzf razor-agents-<ver>.tar.gz
cd razor-agents-<ver>
perl Makefile.PL
make
make install
```

Advanced Mimedefang Rules:

Now that in-bound email is processed by the Mimedefang-filter it is possible to write custom rules in Perl to meet more complex business requirements.

If the Sendmail/Mimedefang server is configured to relay mail for end users then configure Sendmail to use SMTP Authentication and TLS encryption. This will allow a user to authenticate before relaying a message while using TLS encryption to ensure that the username and password, as well as the message itself, is encrypted during transport.

See Claus Aßmann's documentation at http://www.sendmail.org/~ca/email/auth.html and at http://www.sendmail.org/~ca/email/starttls.html for information on how to set up SMTP Authentication or TLS.

SMTP Authentication and TLS provide added relaying security, but also allow custom rules can use the fact that the user authenticated to adjust how the mail system treats the message. Jeremy Mates has some excellent documentation at http://sial.org/howto/mimedefang/macro-pass/, which shows how to pass a Sendmail macro to Mimedefang. Setting a macro with authentication information could be used in an if() statement to omit authenticated users from spam filters or attachment filtering.

It would also be possible to write a rule that rejects all inbound email from the mail domain unless the remote host authenticated. This would eliminate email spoofing from unknown users.

Conclusion:

Installing Sendmail, Mimedefang, Spamassassin, Clamav, Razor, and Anomy Sanitizer greatly reduces the risk of email worm and virus infection by calling three different mime parsers for each inbound message then scanning each MIME part with the clamav virus scanner. Spam is also greatly reduced because each message is checked against multiple spam filtering technologies that are scored together to determine the likelihood of the message being spam. Invalid messages are also rejected causing the remote SMTP host to generate the bounce message to the user cutting down on bandwidth, complexity, and administration.

While those things bring value and control to any mail installation, the greatest benefit is the ability to process mail using Perl without a great performance penalty. Perl processing gives the administrator virtually unlimited message filtering possibilities. Deploying rules that are simply not broad enough in scope to warrant inclusion in commercial products are now trivial to deploy.

Links to Products and Documentation:

Sendmail:	
Homepage	http://www.sendmail.org
FAQ	http://www.sendmail.org/faq/
Example config	http://www.sendmail.org/~ca/
O'Reilly book	http://www.oreilly.com/catalog/sendmail3/
Newsgroup	http://groups.google.com/groups?group=comp.mail.sendmail
Mimedefang:	
Homepage	http://www.mimedefang.org
FAQ	http://www.mimedefang.org/node.php?id=6
Mailing list	http://lists.roaringpenguin.com/pipermail/mimedefang/
Spamassassin:	
Homepage	http://www.spamassassin.org
FAQ	http://wiki.apache.org/spamassassin/FrequentlyAskedQuestions
Documentation	http://spamassassin.apache.org/doc.html
Clamav:	
Homepage	http://www.clamav.net
FAQ	http://www.clamav.net/faq.html#pagestart
Documentation	http://www.clamav.net/doc/
Razor:	
Homepage	http://razor.sourceforge.net
FAQ	http://razor.sourceforge.net/docs/faq.php
Documentation	http://razor.sourceforge.net/docs/
Anomy Sanitizer:	
Homepage	http://mailtools.anomy.net/
Mailing list	http://mailtools.anomy.net/archives/anomy-list/
Documentation	http://mailtools.anomy.net/sanitizer.html
Perl:	
Homepage	http://www.perl.org/
FAQ	http://faq.perl.org/
Documentation	http://www.perl.org/docs.html
O'Reilly book's	http://perl.oreilly.com/

References:

[1] N. Borenstein, Bellcore and N. Freed, Innosoft. “MIME (Multipurpose Internet Mail Extensions).” RFC 1341. June 1992. URL: http://rfc.net/rfc1341.html (25 October 2004).

[2] Dept. of Electrical Engineering, University of Delaware, Newark, DE. “STANDARD FOR THE FORMAT OF ARPA INTERNET TEXT MESSAGES.” RFC822. August 13, 1982.
URL: http://rfc.net/rfc822.html (25 October 2004).

[3] Acohido, Byron and Swartz, Jon. “Are hackers using your PC to spew spam and steal?” USA Today. September 8, 2004. URL: http://www.usatoday.com/tech/news/computersecurity/2004-09-08-zombieuser_x.htm (25 October 2004).

[4] Trend Micro. “Badtrans-A and Badtrans-B worms on Windows.” CERT-IST. November 27, 2001. URL: http://www.cert-ist.com/english/advisories/avis_badtrans_en.htm (25 October 2004).

[5] Microsoft Corporation. “Microsoft Security Bulletin (MS01-020).” TechNet. June 23, 2003. URL: http://www.microsoft.com/technet/security/bulletin/MS01-020.mspx (25 October 2004).

[6] Helenius, Jari. “NAI Webshield SMTP for WinNT MIME header vuln that allows BadTrans to pass.” Bugtraq Mailing List. November 29 2001. URL: http://cert.uni-stuttgart.de/archive/bugtraq/2001/11/msg00267.html (25 October 2004).

[7] Symantec Corporation. “W32.Aliz.Worm.” Symantec Security Response. January 22, 2004. URL: http://www.symantec.com/avcenter/venc/data/w32.aliz.worm.html (25 October 2004).

[8] Link, Rainer and Hecking, Lars. “AMaViS Security Announcement.” August, 26 2002. URL: http://www.amavis.org/security/asa-2001-1.txt (25 October 2004).

[9] Activatormail. “Microsoft Outlook and Mail Server Virus Vulnerabilities.” URL: http://www.activatormail.com/vulnerabilities.htm (25 October 2004).

[10] Refsnes Data. “The <iframe> tag.” URL: http://www.w3schools.com/tags/tag_iframe.asp (25 October 2004).

[11] McLellan, Jason. “A Secure Sendmail Based DMZ for the Corporate Email Environment.” Sans Reading Room. January 13, 2003. URL: http://www.sans.org/rr/papers/19/968.pdf (25 October 2004).

[12] Wikipedia. “Mbox.” Wikipedia. October 21, 2004. URL: http://en.wikipedia.org/wiki/Mbox (25 October 2004).

[13] Tschabitscher, Heinz. “What You Need to Know About Bayesian Spam Filtering.” About.com. URL: http://email.about.com/cs/bayesianfilters/a/bayesian_filter.htm (25 October 2004).

[14] Vipul, Ved Prakash. “Frequently Asked Questions.” Vipul’s Razor Documentation. October 15, 2004. URL: http://razor.sourceforge.net/docs/doc.php?type=text&name=FAQ (25 October 2004).

Last Updated: May 29th, 2017

Upcoming Training

Security Operations Center Summit & Training	Washington, DC	Jun 05, 2017 - Jun 12, 2017	Live Event
SANS Houston 2017	Houston, TX	Jun 05, 2017 - Jun 10, 2017	Live Event
Community SANS Ottawa SEC401	Ottawa, ON	Jun 05, 2017 - Jun 10, 2017	Community SANS
SANS San Francisco Summer 2017	San Francisco, CA	Jun 05, 2017 - Jun 10, 2017	Live Event
SANS Charlotte 2017	Charlotte, NC	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Rocky Mountain 2017 - SEC401: Security Essentials Bootcamp Style	Denver, CO	Jun 12, 2017 - Jun 17, 2017	vLive
SANS Secure Europe 2017	Amsterdam, Netherlands	Jun 12, 2017 - Jun 20, 2017	Live Event
Community SANS Portland SEC401	Portland, OR	Jun 12, 2017 - Jun 17, 2017	Community SANS
SANS Rocky Mountain 2017	Denver, CO	Jun 12, 2017 - Jun 17, 2017	Live Event
SANS Minneapolis 2017	Minneapolis, MN	Jun 19, 2017 - Jun 24, 2017	Live Event
SANS Columbia, MD 2017	Columbia, MD	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS Cyber Defence Canberra 2017	Canberra, Australia	Jun 26, 2017 - Jul 08, 2017	Live Event
SANS Paris 2017	Paris, France	Jun 26, 2017 - Jul 01, 2017	Live Event
SANS London July 2017	London, United Kingdom	Jul 03, 2017 - Jul 08, 2017	Live Event
Cyber Defence Japan 2017	Tokyo, Japan	Jul 05, 2017 - Jul 15, 2017	Live Event
Community SANS Phoenix SEC401	Phoenix, AZ	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Munich Summer 2017	Munich, Germany	Jul 10, 2017 - Jul 15, 2017	Live Event
SANS Cyber Defence Singapore 2017	Singapore, Singapore	Jul 10, 2017 - Jul 15, 2017	Live Event
Community SANS Minneapolis SEC401	Minneapolis, MN	Jul 10, 2017 - Jul 15, 2017	Community SANS
SANS Los Angeles - Long Beach 2017	Long Beach, CA	Jul 10, 2017 - Jul 15, 2017	Live Event
Mentor Session - SEC401	Macon, GA	Jul 12, 2017 - Aug 23, 2017	Mentor
Mentor Session - SEC401	Ventura, CA	Jul 12, 2017 - Sep 13, 2017	Mentor
Community SANS Atlanta SEC401	Atlanta, GA	Jul 17, 2017 - Jul 22, 2017	Community SANS
Community SANS Colorado Springs SEC401	Colorado Springs, CO	Jul 17, 2017 - Jul 22, 2017	Community SANS
SANSFIRE 2017	Washington, DC	Jul 22, 2017 - Jul 29, 2017	Live Event
Community SANS Charleston SEC401	Charleston, SC	Jul 24, 2017 - Jul 29, 2017	Community SANS
SANSFIRE 2017 - SEC401: Security Essentials Bootcamp Style	Washington, DC	Jul 24, 2017 - Jul 29, 2017	vLive
Community SANS Fort Lauderdale SEC401	Fort Lauderdale, FL	Jul 31, 2017 - Aug 05, 2017	Community SANS
SANS San Antonio 2017	San Antonio, TX	Aug 06, 2017 - Aug 11, 2017	Live Event
SANS Prague 2017	Prague, Czech Republic	Aug 07, 2017 - Aug 12, 2017	Live Event
SANS Boston 2017	Boston, MA	Aug 07, 2017 - Aug 12, 2017	Live Event