Defense In Depth:
Anti-SPAM for sendmail Environments

by David Bank, CCSE, CNE, CNA
v1.50 (2007-November-11)

The Need for Effective Anti-SPAM

E-Mail under attack

E-Mail was the Internet's first "killer app". From humble beginnings as a sub-function of FTP, E-Mail has grown into an important part of any network infrastructure. Street addresses and phone numbers disappear in favor of E-Mail addresses. Individuals register "vanity" Domain names. An address in a "cool" E-Mail Domain, like **@gmail.com**, has become a status symbol.

Unlike postal (*aka* snail) mail, E-Mail places the cost burden primarily on the recipient (or the recipient's service provider), not the sender. This economic imbalance has given rise to the phenomenon of SPAM, also called Unsolicited Commercial E-Mail (UCE). In the mid-1990s, SPAM was a relatively minor annoyance. Now, the sheer volume of SPAM is a threat to the viability of E-Mail as a communications medium.

By most estimates, 80% or more of E-Mail is SPAM[1][2][3]. Mail servers face an ever-increasing onslaught of fraudulent connections trying to pound SPAM through to users. SPAM chews up bandwidth, sucks down disk space, and adds to the clutter of the electronic desktop that's now a part of many workplaces. If E-Mail is to remain a viable tool, effective anti-SPAM defenses are a regrettable necessity.

A SPAM-fighting philosophy

The anti-SPAM philosophy of this paper embodies the military concept of "defense in depth", a strategy that seeks to resist, delay and obstruct the attacker (make no mistake, spammers are attackers) with multiple defensive layers, rather than attempting to stop the attacker outright with a single defensive layer. Also known as an "elastic defense", a successful implementation of this concept results in an anti-SPAM defense that is easily adapted to new threats, is difficult for attackers to trick, and minimizes false-positives.

In practice, this paper describes an anti-SPAM defense designed to stop SPAM as early and often as possible, without relying on a single defensive technique. The sooner a connecting host can be identified as a SPAM source, the sooner the connection can be dropped, and the less bandwidth, CPU time and disk space the spammer gets to waste. To this end, this paper's approach relies on pre-acceptance tools to detect the obviously fraudulent connections early in the E-mail process. By eliminating easily-detected SPAM early, resources are conserved for dealing with the "smarter" spammer later in the process.

Some anti-SPAM arrangements rely primarily on post-acceptance tools, such as SpamAssassin. While this paper advocates the use of SpamAssassin, it is engaged very late in the anti-SPAM process. The reasoning is simple: SpamAssassin requires that the E-Mail be transmitted. From the spammer's perspective, the "payload" has been delivered; the victim's bandwidth, CPU and disk space have been successfully wasted. The concept of "defense in depth" is oriented toward stopping as much SPAM as possible before it reaches an "expensive" tool like SpamAssassin.

Because SPAM is also used as a vector for virus propagation, and malware-controlled machines (*aka* 'bots) are used by spammers, anti-Virus (AV) software is considered an important component of anti-SPAM defenses. The configurations presented in this paper include using AV both to defend systems from viruses and to help detect SPAM.

Of course, there are any number of schools of thought as to how to fight SPAM. This paper discusses one particular school; if you find that it doesn't work for you, develop or adopt a different school of thought. The purpose of this paper is not to gain converts, but to present software and techniques found to provide reasonably effective[4] defenses at a reasonable (very low) cost. This is not to say that other software combinations or techniques are not "effective". The Alternatives section, below, offers additional ideas.

Finally, the Electronic Frontier Foundation has published a white paper entitled *Noncommercial Email Lists: Collateral Damage in the Fight Against Spam*[5]. Whatever your SPAM-fighting philosophy, you should consider the issues presented in that paper, and tailor your strategy appropriately. Of particular importance, have a viable method for legitimate senders to reach you and request whitelisting; additionally, you should have a mechanism for your users to request consideration for specific senders.

RBLs: Good or Evil?

Along with many schools of thought for anti-SPAM, there are schools of thought about Real-time Blackhole Lists, or **RBL**s. In brief, an RBL is a list of hosts/Domains that have been identified, by the organization hosting the RBL, as a source of SPAM.

Some ardent SPAM fighters decry RBLs as a blunt instrument, a tool that is clumsy at best and dangerous at worst. There have been examples of "legitimate" sites/Domains/ISPs finding themselves listed on a popular RBL, much to the dismay of their users/customers. It is true that RBLs can provide bad information.

However, an RBL is merely a tool, one of many available to combat SPAM. You, as the mail system administrator, have the power to decide how you will use the information from the RBL. In the simplest approach, you can reject any RBLed host; this is frequently how RBLs are used, and the main reason their use is decried by some in the anti-SPAM community. Alternatively, you can leverage other tools that will let you use RBLs as just one input in the anti-SPAM process.

Therefore, RBLs are neither good nor evil. It's how they are used that makes the difference.

Tool selection

This paper presents anti-SPAM defenses that rely on three Free/Open Source Software (FOSS) packages: MIMEDefang[6], SpamAssassin[7] and ClamAV[8]. All of these integrate with the ubiquitous sendmail[9] mail server software, which is also a FOSS package.

These software choices basically require use of a Linux or UNIX system as the underlying OS for the configurations presented in this paper. Although ports of these softwares exist, enabling you to run them on other platforms, the necessary tweaks or adjustments are outside the scope of this paper.

As of this writing, the current versions of these tools are:

sendmail v8.14.2
MIMEDefang v2.63
SpamAssassin v3.2.3
ClamAV 0.91.2

The details of building/installing these softwares, and their general configuration, are outside the scope of this paper. However, if you build the tools from source, the **How to Build Your Defenses** section below touches on some build options specific to the configurations in this paper. If you use pre-compiled packages, check the build options used by the maintainer.

What is a **MILTER**?

The configurations presented in this paper rely upon the **MILTER** functionality of sendmail to create a cohesive anti-SPAM system. A sendmail MILTER is an external program (MIMEDefang, in this paper) that participates in the SMTP conversation between sendmail and the connecting mailhost.

SMTP conversations proceed in steps: the initial connection, **`HELO`**, **`MAIL FROM:`**, **`RCPT TO:`** and **`DATA`** are the primary steps with which we are concerned. Anti-SPAM defenses can be engaged at each step. During each step, sendmail applies whatever logic it is configured to use (for example, during the initial connection step, **`Connect:`** entries in the sendmail *access* table, if any match, are applied). Then, assuming sendmail has decided to allow the connection to proceed, the information sent by the connecting host is passed to the MILTER, which may perform its own analysis and return an action to sendmail. While multiple MILTERs may be used, each being engaged in series, this paper will consider a single-MILTER design.

If you don't yet understand the basics of MILTERs, it would be a good idea to read up on them before proceeding with the configurations discussed in this paper[10][11]. It is important that MIMEDefang is compiled using the **libmilter** library appropriate to the version of sendmail with which it will communicate.

Alternatives

As mentioned above, there are many anti-SPAM tools. This paper covers some that are effective for particular situations. Your mileage, as they say, may vary. Other tools out there include:

Vipul's Razor[12]
Distributed Checksum Clearinghouse (DCC)[13]
Pyzor[14]
SnertSoft's sendmail MILTERs[15]
SpamBayes[16]

Many of these tools can be integrated with MIMEDefang and augment the configurations shown in this paper. Some tools function best in a stand-alone manner. Choose the tools that best meet your needs.

How to Build Your Defenses

This paper covers an anti-SPAM defense for a mail environment that uses a mail relay at its border. This mail relay doesn't host user mailboxes; instead, it is a single point of entry/exit for all E-Mail entering or leaving the network. While a singular mail relay is discussed, the configurations presented are equally applicable for a clustered or round-robin environment where multiple machines share the mail handling load. The design is also easily extended to a scenario where such a relay handles E-Mail for multiple Domains.

Splitting mail routing apart from mailbox hosting has several advantages that outweigh the increase in complexity, in most situations. First, users attempting to access their mailboxes don't have to compete with the spammers for the server's attention. Next, the relay host can be hardened to a high degree, exposing only minimal ports to the outside world. Finally, some anti-SPAM techniques work better when the E-Mail is not destined for a mailbox hosted on the local server.

Standards and Assumptions

This paper makes a number of assumptions about the target environment, including the use of a modern Linux or UNIX (or UNIX-like) operating system, and that the software tools (see Tool selection above) are already installed and working. As noted above, installation of the tools is outside of the scope of this document.

Instead, this paper focuses on configuration of those tools with a specific goal of fighting SPAM. Since the options selected at compile-time can affect the ability of these tools to help in the fight, some important ones will be mentioned. Note that SpamAssassin is installed as a Perl module, and so does not have compilation options[17]. Similarly, sendmail v8.13.x and later has MILTER support enabled by default and so its options are omitted.

ClamAV: SPAM-fighting Compilation Options

`--with-zlib=`*/path/to/zlib*

This option is mentioned to emphasize that support for **zlib** is necessary if you want **clamd** to be able to scan compressed file attachments. Scanning compressed file attachments is important to prevent malware from slipping by in a compressed form.

`--with-user=`*clam*

Using this option, you can specify a non-privileged user ID under which the **clamd** and **freshclam** daemons will run. The default value is *clam*. This user name must exist or the daemons will not run. This option is important as a security measure; while **clamd** is launched as *root*, never configure it to run as a privileged user. This option can be overridden with the **User** statement in the *clamd.conf* file (discussed below).

`--with-group=`*clam*

Similar to the previous option, you can specify the Group ID under which the **clamd** and **freshclam** daemons will run. The default value is *clam*. This Group name must exist or the daemons will not run. Again, do not allow the daemons to run as an inappropriate Group.

`--enable-bigstack`

Spammers are perfectly happy to attack your systems in order to SPAM them. This option helps **clamd** resist stack overflow attacks by altering the program code. The result is a program stack size 64KB larger that whatever is specified as the standard stack size for the operating system (usually defined via */usr/include/stdio.h*). RAM is relatively cheap, and enabling this option is a fairly inexpensive form of insurance against attacks that depend on triggering excessive recursion in **clamd**.

`--without-libcurl`

curl is a FOSS package for programmatically accessing data using URLs. Within ClamAV, it is used to check URLs in E-Mail messages/attachments for malware. The goal is to thwart the malware author who relies on gullible users following the link in a SPAM E-Mail.

A perfectly good software package, the goal of **curl** is laudable. If, in your environment, leveraging ClamAV in this fashion is appropriate, then by all means enable **curl** (assuming you have it installed already). Within the configuration presented in this paper, **curl** is not used for several reasons – it's regarded as the job of the local system's A/V scanner to detect malware downloads from web pages, the resources necessary for a mail relay to perform this task are potentially excessive, and such investigations open up DoS or overflow attacks against the relay.

MIMEDefang: SPAM-fighting Compilation Options

`--with-sendmail=`*/usr/lib/sendmail*

Included mainly as a reminder that this paper is geared towards a configuration using modern **sendmail** as the MTA. Remember that MIMEDefang requires the use of a Queue Runner (MSP) instance of **sendmail**. Integration with modern versions of PostFix[18] may be possible, but are outside the scope of this paper.

`--with-user=`*defang*

This option defines the User ID under which MIMEDefang will run. The default is *defang*. The daemons launch as *root*, then switch user context to this ID. You should not allow the daemons to run as a privileged user ID.

`--enable-debugging`

Using this option causes the MIMEDefang Multiplexor to add debugging messages to its debug file, which is hard coded as */var/log/mdefang-event-debug.log*. The file must exist or the messages will not be written. This option is not required for the configurations presented in this paper, but may be useful in troubleshooting. It is important to not confuse this compilation option, which only affects the Multiplexor, with the **`-d`** command-line switch on the **mimedefang** executable.

Configuring Your SPAM Fighting Arsenal

We'll look at each individual component, starting with the run-time configuration options to which you should pay particular attention.

ClamAV: Configuration File

In the design proposed in this paper, **clamd** is set up to interface via a UNIX socket. Using a socket, any program can communicate with **clamd**, but this configuration will focus on MIMEDefang.

The **clamd** daemon relies exclusively on its configuration file, and does not accept command-line configuration parameters. However, you can specify the path and name of the configuration file using the **`-c`** command-line parameter. Absent that, **clamd** looks for *clamd.conf* in whatever path that was defined during compilation (*c.f.* the **sysconfdir** compilation option).

The default configuration file is both complete and well-documented. This paper will focus on those configuration settings that are important to the interface with **MIMEDefang**. Configuration settings with BOOLEAN (*e.g.* on/off, true/false, yes/no) values are set by commenting/uncommenting the entry in *clamd.conf*.

`LogSyslog`

By default, **clamd** does not perform any logging. Uncommenting this entry enables **clamd** to log to the **syslogd** daemon. The **syslogd** daemon must be running, and must be configured to accept log messages for the appropriate *Facility*.

`LogFacility` ***LOG_LOCAL5***

Unless this entry is used to specify a different *Facility*, a syslog-enabled **clamd** will log to *Facility* **LOCAL6** by default. The parameter of this entry uses the syntax **LOG_<*Facility*>**; in this example the *Facility* is changed from the default to **LOCAL5**.

`LogVerbose`

If you find **clamd**'s log entries too laconic, you may increase the verbosity by uncommenting this entry.

`PidFile` ***/var/run/clamd.pid***

In its default configuration, **clamd** does not create a Process ID, or PID, file. This entry allows you to specify a path and a filename. Note that this location must be writable by

the User ID under which the daemon runs (however, since **clamd** is started as *root*, the file does not need to exist or have particular ownership/mode beforehand). Having **clamd** create a PID file is very useful for ensuring correct start-up and shut down sequences.

`LocalSocket` ***/var/spool/MIMEDefang/clamd.sock***

clamd can interface with a local or a network socket. The local socket is more secure; the network socket allows other hosts to submit data streams to **clamd** for scanning. In the configuration presented in this paper, the local socket is used for the MIMEDefang connection. The value of this entry should be the full path and filename of the socket.

`FixStaleSocket`

You should uncomment this entry, which enables the ability of **clamd** to clean up a stale (unused) socket in the event of a system crash or other problem that leaves the socket file in place when **clamd** isn't running. If you don't enable this, and something happens to cause **clamd** to terminate but not clean up its socket, then **clamd** will be unable to restart until the socket file is removed.

`StreamMaxLength 2M`

This entry defines the maximum size of a data stream that **clamd** will attempt to scan for viruses. If this limit is reached, **clamd** will terminate the connection. The value is a positive integer followed by a unit specifier, which may be either **`K`** or **`M`**, for *kilobytes* or *megabytes*, respectively. The specifier is case-insensitive, so **`k`** or **`m`** may also be used.

The default value is **`10M`**; however, you should change this to match the maximum message size that your MTA will accept (sendmail uses the **`confMAX_MESSAGE_SIZE`** macro to determine this, and is not limited by default). You can also configure MIMEDefang to not attempt submission of messages that exceed this size (code to do that is shown in the default filter file).

`SelfCheck 3600`

This configuration entry specifies the interval between **clamd** checks of its database integrity and freshness. The value is read in seconds and defaults to **1800** (30 minutes). Here, we increase it to **3600**, or one check per hour. If you prefer a more or less frequent interval, change this accordingly.

`User defang`

Without specifying a User ID under which **clamd** should run (either here or at compile-time), it will execute as *root* (although the daemon will give up its special privileges once it is running). To aid integration with MIMEDefang, as is the configuration presented in this paper, you need to have **clamd** run as the same User ID as MIMEDefang (whatever that may be).

`AllowSupplementaryGroups`

By default, when context is switched from *root* to the User ID specified in the previous entry, the Secondary Groups are not included in the security context. The Group ID in the */etc/passwd* entry is the User ID's Primary Group; any additional Group IDs listed in */etc/group* are Secondary Groups. Uncommenting this entry allows the **clamd** daemon to claim the security privileges of the Secondary Groups, and is needed in this configuration to allow the interface with MIMEDefang to work properly.

`ArchiveMaxFileSize 1M`

This entry specifies a limit on file sizes extracted from archives (not the archive itself). If a file extracted from an archive is over the limit, it will not be scanned. This helps avoid a DoS attack that uses huge attachments, each compressed and stored within another compressed file. Similar to the **`StreamMaxLength`** entry, the value is a positive integer followed by unit specifier. The default value is **`10M`**, and you should adjust this as your system requires. This example specifies a fairly low limit.

Primarily, you want **clamd** to be a resource for MIMEDefang. The job of **clamd** is to scan data streams sent by MIMEDefang and report back on the detection of viruses. The “brains” of our arrangement are mostly located in the MIMEDefang configuration, and you're asking **clamd** to be thorough, but not “smart”.

SpamAssassin: Configuration File

Within the configuration presented in this paper, the usual SpamAssassin configuration methods are not used. Instead, there is a special version of the configuration file, *sa-mimedefang.cf*, located in the MIMEDefang configuration directory. A sample of this is in the **[Reference]** section below.

It is crucial to understand that, in the configuration presented, SpamAssassin cannot make any changes to the message headers or body. That is, you may instruct SpamAssassin to change those things, but the changes will not actually propagate back through MIMEDefang.

Similar to ClamAV, the system design presented in the paper regards SpamAssassin as a resource for MIMEDefang. The job of SpamAssassin is to evaluate E-Mails submitted by MIMEDefang using its rulesets, and issue a judgment on the "spamminess" of the E-Mail. The ultimate decision of how to react to that evaluation is made by MIMEDefang.

With respect to the particular configurations of this paper, there are some entries in *sa-mimedefang.cf* that are important:

```
rewrite_subject 0
```

As discussed above, no changes to the E-Mail made by SpamAssassin will actually take effect. You can save the occasional CPU cycle by telling SpamAssassin not to bother with rewriting the **Subject:** header.

```
report_header 1
```

While SpamAssassin cannot modify the E-Mail, it can generate a report that will be passed back to MIMEDefang. So it's OK to turn this on.

```
trusted_networks 127.0.0.1
```

Don't forget to tell SpamAssassin to trust your network, or mail hosts, as appropriate to your environment. This is actually a catch-all, as the MIMEDefang configuration presented in this paper suggests not submitting your "internal" E-Mail to SpamAssassin.

```
defang_mime 0
```

Again, since SpamAssassin is unable to make lasting changes to the E-mail, there's no point in it wasting time fiddling with MIME headers.

MIMEDefang: Configuration File

The system design outlined in this paper makes MIMEDefang the centerpiece of the anti-SPAM defenses. Hence, its configuration is the most complex part of this paper, and the process is made more involved by the fact that even a minimal MIMEDefang system has a relatively complex configuration file.

Unlike the other components, the MIMEDefang configuration file is actually part of the program itself. It's written in Perl, and at least a passing understanding of Perl is necessary to modify the configuration. The file is named *mimedefang-filter* and is stored in the MIMEDefang configuration directory.

In the discussion of MILTERs, it was noted that the SMTP conversation proceeds in steps, and that the MILTER is involved at each step. In the simplest configuration, a MILTER would do nothing and merely return processing control back to sendmail. This is what happens when a MIMEDefang filter file defines no filtering function for a particular step.

In its default configuration, MIMEDefang only performs filtering at the **DATA** step in the SMTP conversation. By invoking the **mimedefang** executable with certain parameters, and adding appropriate functions to your *mimedefang-filter* file, you can engage MIMEDefang earlier in the SMTP conversation and perform a number of anti-SPAM activities. This table shows which function names in *mimedefang-filter* are invoked at which step in the SMTP conversation, and also what command-line parameter is needed on the **mimedefang** invocation:

SMTP step	*mimedefang-filter* function(s)	**mimedefang** invocation command-line parameter
initial connection	`filter_relay`	`-r`
HELO	`filter_helo`	`-H`
MAIL FROM:	`filter_sender`	`-s`
RCPT TO:	`filter_recipient`	`-t`
DATA	`filter_begin, filter, filter_multipart, filter_end`	N/A

The contents of *mimedefang-filter* are discussed in more detail below.

Putting It All Together

This paper discusses a design that has interlocking components, and these create dependencies. When fully implemented, MIMEDefang cannot function without **clamd** already running on the system. Similarly, sendmail will be unable to run without MIMEDefang's daemons also already running. You should insure that your system startup (and shutdown) procedures account for these dependencies.

MIMEDefang defines a number of "background" functions in the standard *mimedefang-filter* file. You should read about and understand these functions, but doing so is not necessary to implement the configurations from this paper. The function names are **`filter_bad_filename()`** and **`defang_warning()`**.

You should also read and understand the documentation in the sample *mimedefang-filter* file, and the **mimedefang-filter (5)** and the **mimedefang (8)** man pages. In particular, pay attention to the parameters given to, and required back from, each function. The code examples below assume you have studied the documentation and understand how to properly invoke and return the functions used.

The default file also only defines the **`filter*`** functions associated with the **DATA** step of the SMTP conversation. This paper will start by adding the additional **`filter_*`** functions needed for the other steps, then re-visiting the **DATA** step. Remember, to enable these additional functions, **mimedefang** must be invoked with the related command-line parameter, and the associated function must be defined in *mimedefang-filter*.

About the Perl example code

Perl is a complex and still-evolving language. It lacks a formal specification, meaning that the only thing defining Perl is the **perl** interpreter itself. As many Perl programmers know, there's always a different way to code a given algorithm in Perl.

The code examples below are not the only way to code the ideas presented; they may not even be the "best" way to code them in Perl. The objective of the code examples in this paper is not to produce the tersest or even most efficient Perl code. Rather, the goal is to clearly delineate the programming logic using syntactically valid Perl (as opposed to, say, pseudo-code). This allows readers with minimal Perl skills to understand and implement the code, while more experienced Perl coders are free to re-write the Perl code in their own unique idiom.

`filter_relay`

When **mimedefang** is invoked with `-r` parameter, and this function is defined, you can use MIMEDefang to augment sendmail's connection filtering. This function is referenced after sendmail has completed any RBL checks, unless sendmail's `FEATURE(delay_checks)` is in use, in which case this function is referenced before sendmail actually performs its RBL checks. sendmail will check and apply any **Connect:** entries in the *access* table before MIMEDefang is passed the relay information.

While you can still use the *access* table for other functions, such as `GreetPause`, `ClientConn` and `ClientRate`, the best advantage to `filter_relay()` is that you can create a finer-grained RBL checking than is offered by sendmail. The trouble with sendmail's RBL functions is that the first RBL hit causes a **Reject**. This leads to the "blunt instrument" charge outlined in RBLs: Good or Evil? above.

Below is some sample code to implement a relay filter. This code is designed to check the IP address of the connecting relay against a set of RBLs, much the way that sendmail's RBL functions would do. The difference is that you can control which, and how many, RBL hits actually result in a rejection of the connection. You can also distinguish a lookup failure from an RBL hit.

The code first checks that the IP address of the connecting host is not in a hash of "internal" host addresses; this prevents your filter from wasting resources on your own hosts, based on the assumption they are "trusted". If the IP doesn't match an internal host, then several blacklist services are checked. The code presented has a timeout of 8 seconds, and the RBL check will exit early if 3 or more RBLs report a match. At least 3 of the RBLs must "hit" or the connection will not be rejected – unless there are 2 hits and one or more RBL server checks failed.

```
# GLOBAL VARIABLES – declared outside any function

##############################
# Hash of internal hosts
#     - Key: IP addresses we consider internal
#     - Value: Flag as to if host if Virus-scan exempt (0=No, 1=Yes)
##############################
%OurHosts=( "127.0.0.1", 0,
            “10.0.0.1”, 0,
            “192.168.1.1”, 1 );

# List of RBL servers to check
@RBL_list=qw{ sbl.spamhaus.org dnsbl.njabl.org bl.spamcop.net
            cbl.abuseat.org };

# Timeout (in seconds) for RBL check
$RBL_timeout=8;
# Maximum number of positive RBL responses before we don't care any more
$RBL_max=3;

[...]

sub filter_relay
{
# Read parameters passed to function
my($ip, $name)=@_;

# Pointer to hash returned from RBL check function
my($rblhash);

# Local variables for analysis
my($rblserver, $rblresult, $rblscore);
my($tempfail_flag)=0;

# Search the hosh of our hosts using the $hostip argument
if ( exists($OurHosts{$ip}) )
      {
      # The connecting host is our own host, don't bother checking further
      return('CONTINUE', 'ok');
      }
else
      {
      # This host is not our host – check RBLs
      $rblhash=relay_is_blacklisted_multi($ip, $RBL_timeout, $RBL_max, @RBL_list);
      }

# Evaluate RBL results
foreach $rblserver (keys(%$rblhash))
      {
      $rblresult=$rblhash->{$rblserver};
      # If the value returned by a specific RBL server is an array, then
      #     the RBL had a listing for this IP
      if (ref($rblresult) eq “ARRAY”)
            {
            $rblscore=$rblscore + 1;
```

```
                }
        else
                {
                if ($rblresult eq "SERVFAIL")
                        {
                        # A lookup failed – set a flag to TEMPFAIL the mail if
                        #     enough other RBLs have the IP listed
                        $tempfail_flag=1;
                        }
                        # End of IF
                }
                # End if IF
        }
        # End of FOR loop
# If the RBL score is RBL_max or higher, we REJECT the connection
if ($rblscore >= $RBL_max)
        {
        return('REJECT', "$name appears on multiple IP blacklists; see SPAM
                database lookup at http://www.dnsstuff.com");
        }
# If we are here, then the IP was not on a sufficient number of RBLs to be
#     rejected. However, we can TEMPFAIL the connection if the IP was listed
#     on "RBL_max - 1" RBLs and there was at least one RBL lookup failure
if ( ($rblscore + $tempfail_flag) == $RBL_max )
        {
        return('TEMPFAIL', "Please try again later");
        }
# If we have reached this point, the connection is either not blacklisted, or
#     does not appear on enough blacklists – allow it to proceed
return('CONTINUE', 'ok');
}
# End of sub filter_relay
```

The benefit of this code design is that, unlike the sendmail RBL functions, no single RBL can cause an E-Mail to be rejected. However, there are some downsides:

1) Your DNS server will be working harder; all RBLs will be queried, rather than the query process stopping at the first positive response.
2) RBLed senders who are legitimate will have no way to reach your postmaster to request whitelisting. The requirement that multiple RBLs show a listing for an given IP hopefully prevents accidental rejections. Alternatively, you could implement this RBL check code in **`filter_recipient()`**, thereby emulating sendmail's **`FEATURE(delay_checks)`**.

filter_helo

This function, available when **mimedefang** is invoked with **-H** parameter, allows you to perform filtering of connections based on the identification provided during the **HELO** step of the SMTP conversation. Prior to sendmail v8.14.0, sendmail performed no effective filtering[19] at this point in the conversation, although using the **GreetPause** feature (available in sendmail v8.13.0 and later) will cause sendmail to drop a host that attempts to send anything prior to your sendmail host presenting its greeting banner.

The example code below checks to see if the connecting host is trusted, and returns an "OK to proceed" result immediately if it is, thus bypassing all the other checks. If the host is not trusted, the code examines the **HELO** string provided by the remote host by first seeing if it is an IP address. If the **HELO** was an IP, it is checked for surrounding square brackets, and to see if it matches the actual IP of the connecting host. If the **HELO** was not an IP address, it's checked for indications that it is an FQDN, and that it is not, or doesn't contain, the string "localhost". The global variables introduced in **filter_relay()** are also used in this function.

```
sub filter_helo
{
# Read parameters passed to function
my($ip, $helo)=@_;

# Search the hosh of our hosts using the $ip argument
if ( exists($OurHosts{$ip}) )
      {
      # The connecting host is our own host, don't bother checking further
      return('CONTINUE', 'ok');
      }

# The connecting host is foreign, examine its HELO for fraud
if ($helo =~ /^(\[?)(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})(\]?)$/ )
      {
      # HELO looks like an IP – the comparison will split the string
      #     into 3 variables; $1 will have [ or be undefined, $2 will
      #     have the IP address without any brackets; $3 will have
      #     ] or be undefined

      # The IP address portion should *not* be identical to the original
      #     HELO string – if it is, the original HELO lacked brackets
      if ( $2 eq $helo )
            {
            # Reject connection – invalid HELO
            return('REJECT', "$helo is not a valid HELO");
            }
      # Since the HELO was an IP address, it should match the IP of
      #     the connecting host
      if ( $2 ne $ip )
            {
            # HELO does not match actual IP – fraudulent HELO
            return('REJECT', "FRAUDULENT HELO: $helo is not $ip");
```

```
            }
            # End of IF
      }
else
      {
      # HELO looks like a host name string

      # If the HELO is an FQDN, it will contain a “.”
      if ( index($helo, '.') == -1 )
            {
            # HELO is not an FQDN
            return ('REJECT', "INVALID HELO: $helo not FQDN”);
            }
      # HELO should not contain “localhost”
      if ($helo =~ /localhost/i)
            {
            # The HELO contains “localhost”
            return('REJECT', "INVALID HELO: $helo not valid identification");
            }
            # End of IF
      }
      # End of IF

# If we got to here, the HELO was reasonable
return('CONTINUE', 'ok');
}
# End of sub filter_helo
```

Rejecting obviously fraudulent (or, more charitably, invalid) **HELO** strings is an effective anti-SPAM measure. You'll see a significant drop in SPAM making it through your defenses by implementing these simple tests. And by dropping such obviously fraudulent **HELO**s, you'll prevent spammers from consuming more than a small amount of your resources.

Note that the FQDN check in the code above is somewhat simplistic, and it would not be terribly difficult to fool it. Spammers are notoriously lazy individuals, however, and so the check is (as of this writing) fairly effective. Possible improvements, left as an exercise to the reader, include:

1) Checking that **$helo** does not claim to be a machine in a Domain being hosted - the only hosts using your Domain(s) in their **HELO** string should be your own hosts, not a foreign host - see **filter_sender()** below for a similar code example
2) Checking the TLD portion of **$helo** for a valid TLD; for example, **host.domain.arg** could be rejected since ***arg*** is not a valid TLD
3) Checking the SPF records for the Domain to verify that **$ip** is a permitted mailhost for the Domain – Perl modules are available from CPAN to do this
4) Attempting to resolve **$helo** through DNS and, if no entry is found, issuing a **REJECT** (be sure to account for a **SERVFAIL** result from DNS, perhaps by a **TEMPFAIL** of the E-mail)

filter_sender

Invoking **mimedefang** with **-s** parameter, and including this function in your filter, allows you to perform checking at the **MAIL FROM:** step of the SMTP conversation. sendmail can perform some filtering here, mainly based on **From:** entries in the *access* table. MIMEDefang offers much more granular control.

While spammers frequently forge the source E-Mail address of their SPAM, a good use of this function is to code a filter of any **MAIL FROM:** claiming to be an address in your Domain(s) but originating from a foreign mail server. This code example builds on the one from **filter_relay()**, and adds an additional global variable:

```
################################
# Declare a hash of hosted Domains
#     - Key: Domain Names we host
#     - Value: A flag as to if the Domain should be
#                 receiving E-Mail (0=No, 1=Yes)
################################
%OurDomains=(     "domain1.tld", 1,
                  “domain2.tld”, 0 };

[...]

sub filter_sender
{
# Read parameters passed to function
my($sender, $ip, $hostname, $helo)=@_;

# Local variables
my(@chksender);
my($chk, $chksenderdomain);

# Search the hosh of our hosts using the $ip argument
if ( exists($OurHosts{$ip}) )
      {
      # The connecting host is our own host, don't bother checking further
      return('CONTINUE', 'ok');
      }

# The sender cannot claim to be a user in any Domain we host since we have a
#     already eliminated the sending host as being ours

# Make sure the address we're checking is in all lower-case
$chk = lc($sender);
# Remove any angle-brackets from address
$chk =~ s/^<//;
$chk =~ s/>$//;
# Split the Sender address into Address and Domain Name
@chksender=split(/\@/, $chk);
# Extract just the Domain
$chksenderdomain=$chksender[1];
```

```
if ( exists($OurDomains{$chksenderdomain}) )
      {
      # Reject connection – fraudulent Foreign sender claiming to be local
      return('REJECT', "FRAUDULENT ADDRESS: $helo is not $chksenderdomain");
      }
      # End of IF
}
# End of filter_sender
```

Filtering on the sender address can be problematic when dealing with legitimate users connecting from remote networks. The MIMEDefang documentation demonstrates how **`filter_sender()`** can check the **`SMTP AUTH`** status of the sending host and allow authenticated senders to proceed.

While that is a reasonable approach, it is not used in this paper. Instead, the author prefers a solution where the legitimate sender connects to an internal mail host protected by a VPN, or uses something like SSH Port Forwarding, to insure that access to mail relay services is limited to authenticated senders without bogging down the sendmail relay handling authentication.

`filter_recipient`

Available when **mimedefang** is invoked with **`-t`** parameter, this function allows you to perform filtering of connections at the **`RCPT TO:`** step of the SMTP conversation. This filter is invoked with each **`RCPT TO:`** issued by the connecting host, and you should code your filter with that in mind.

As noted in **`filter_relay()`**, placing the RBL checks in **`filter_recipient()`** is equivalent to **`FEATURE(delay_checks)`** in sendmail, and allows you to create an address to which even RBLed senders can write to request whitelisting. Unless you have very loose rules on when you reject E-Mail from blacklisted hosts, having this sort of "escape hatch" is important.

Alternatively, if you performed the RBL checks earlier and simply maintained that information, then you could reference the RBL results in **`filter_recipient()`** without having to perform the RBL checks for each **`RCPT TO:`**. See the **mimedefang-filter** (**5**) man page entry concerning maintaining state between functions for very important information about this technique; you would have to perform the RBL checks in **`filter_helo()`** or **`filter_sender()`**, as it's not possible to preserve state information gathered in **`filter_relay()`**.

Because the system design presented in the paper calls for MIMEDefang to run on a mail relay, with no local user mailboxes, there's no way (in the stock configuration) for either sendmail or MIMEDefang to know if a given recipient actually exists on the server to which the E-Mail will eventually be delivered. Spammers often launch "dictionary spam", sending E-mail to all possible addresses in a Domain. Without some way to validate a recipient address, your mail relay could end up generating a lot of bounce messages as final delivery of the E-Mail fails. Since spammers typically forge the **From:** address, your mail relay may find itself trying to send bounce messages to non-existent hosts/Domains, or to some poor end-user whose E-Mail address was used in the forged **From:** header.

To help avoid this, you can code **`filter_recipient()`** to check the validity of each recipient using MIMEDefang's built-in function **`md_check_against_smtp_server()`**, which can verify the presence of an address on a target server (assuming that target server supports the operation by responding with the appropriate failure message for non-existent addresses during **`RCPT TO:`**). You should read and understand the **mimedefang-filter(5)** man page entry concerning *Rejecting Unknown Users Early* before implementing filter code like this example:

```
sub filter_recipient()
{
# Read the parameters passed to the function
my($recipient, $sender, $hostip, $hostname, $first, $helo, $rcpt_mailer,
            $rcpt_host, $rcpt_addr) = @_;

# Local variable to hold results of md_check_against_smtp_server
my(@chk_result);

# A HELO string for MIMEDefang to use so that log entries on the other
#     mailserver can be distinguished as MIMEDefang checking
my($chkhelo)=”mimedefang.ourdomain.tld”;

# Search the hosh of our hosts using the $hostip argument
if ( exists($OurHosts{$hostip}) )
      {
      # The connecting host is our own host, don't bother checking further
      return('CONTINUE', 'ok');
      }
else
      {
      # Verify the recipient address on $rcpt_host is valid using
      #     md_check_against_smtp_server
      @chk_result = md_check_against_smtp_server($sender, $recipient, $chkhelo,
                        $rcpt_host);
      }

# Return the result without interpretation
return(@chk_result);
}
# End of sub filter_recipient
```

An obvious improvement to the example code, left as an exercise for the reader, would be to interpret the results in more detail, and have program logic to make decisions based on those results. Because **`filter_recipient()`** is called for each **`RCPT TO:`** individually, tracking the number of failures racked up by a specific connection would require recording state information. If you did that, however, you would be able to set a limit on the number of failed recipient addresses you would accept before doing something more drastic than rejecting an individual address (for example, adding the connecting host to your own internal blacklist, or reporting it to your favorite RBL service)[20].

The Default MIMEDefang Filter

The *mimedefang-filter* file included with MIMEDefang lays out a good, if basic, message filter. It is composed of a set of global variables, followed by the default set of functions: **`filter_bad_filenames()`**, **`filter_begin()`**, **`filter()`**, **`filter_multipart()`**, **`defang_warning()`** and **`filter_end()`**.

At a minimum, you need to change the global variable **`$DaemonAddress`** to a suitable E-Mail address. The global **`$MaxMIMEParts`** is commented out by default. You should enable the limit by uncommenting it, and can also change the value. If set to a positive integer, then messages with more than that number of MIME parts will be bounced at the **`DATA`** step. The goal of this setting is to foil DoS attacks based on handing your mail server an E-Mail with an impossibly huge number of MIME parts. When enabling this protection, use the value provided in the default filter, or perhaps even a lower limit.

The global **`$md_graphdefang_log_enable`** is uncommented, defines the logging *Facility* as **MAIL** and specifies summary logging output. Using this feature is optional, but consider directing the output to a different *Facility*. Comment it out if you want to disable these log entries.

You can also add your own global variables, and the code examples above rely on some added global variables. Beyond those in the examples, consider adding a **`$FilterVersion`** variable, for use in your logging statements. This would allow you to associate log entries to a particular revision of your filter. For example (matching the version of this paper):

```
$FilterVersion=”1.50”;
```

If you decide to use your own logging statements, via the **`md_syslog()`** function, to augment or replace the **`md_graphdefang_log()`** function used by default, then be sure to set the global **`$SyslogFacility`** to the proper **syslogd** *Facility*, like so:

```
$SyslogFacility=”local3”;
```

The example filter associated with this paper makes extensive use of the **`md_syslog`** function to track filter progress and program flow. This is done for illustrative purposes, and you may not wish to have such extensive logging in a production filter. However, detailed logging can help ensure that your filter is performing as expected[21].

In this paper's configuration, the **clamd** AV scanner is used, so we can simply instruct MIMEDefang to use that feature with the globals:

```
$Features{'Virus:CLAMD'} = 1;
$ClamdSock = "/var/spool/defang/MIMEDefang/clamd.sock";
```

Additionally, SpamAssassin is used, you can go ahead and set specific globals for it. Define the path to a MIMEDefang version of the SpamAssassin configuration file like so:

```
$path_sa_conf = "/path/to/sa-mimedefang.cf";
```

Because SpamAssassin can bog down with very large messages, you can set a size limit global for it, and then check the size of an E-mail against this value prior to calling SpamAssassin. This example will limit SpamAssassin scans to messages that are 100KB or smaller:

```
$SAScanSizeLimit = 100 * 1024;
```

Finally, you can initialize SpamAssassin as part of your global setup:

```
spam_assassin_init($path_sa_conf);
spam_assassin_init()->compile_now(1);
```

Now you're ready to start with the core filtering functions. Because sample functions are included in the default MIMEDefang package, code examples will not be shown here. Instead, reference the default filter file included in MIMEDefang.

`filter_begin`

This function marks the entry point to the core MIMEDefang E-Mail filter, and is called following the **`DATA`** step of the SMTP conversation, after the E-Mail has been received and stored in a queue. In addition to all the information about the sending host, you also have the full message headers and access to a wide array of sendmail macros. Unlike the earlier functions, global variables set in **`filter_begin()`** persist through **`filter_end()`**.

The default version of this function is very basic. It examines the message headers for suspicious characters, then submits the E-Mail to any virus scanners that were found when MIMEDefang initialized, using the generic **`message_contains_virus()`** function. If suspicious characters or a virus are found, the message is discarded; if the virus scanner has a temporary failure, then the message is **TEMPFAIL**ed; otherwise, the message is allowed to proceed.

One obvious change to **`filter_begin()`** would be to bypass some or all of the default checks for your own internal hosts. The previous code examples demonstrate code to check the connecting host IP against a list of IPs, and in **`filter_begin()`** the global variable **`$RelayAddr`** contains the connecting host IP. Since that is a global variable, it is also available in **`filter()`**, **`filter_end()`** and **`filter_multipart()`**. The choice to exempt your internal hosts from header checks or virus scanning is as much a business as a technology decision, and is specific to your organization.

If a message contains a virus, you can choose to accept the message, strip out the virused part in **`filter()`** or **`filter_multipart()`**, and allow the text to proceed, with or without a notification

of the alteration. If you allow the message to proceed, you can also use the fact that the message contained a virus to enhance the SPAM score result from SpamAssassin. The base **`filter_begin()`** code shows a flag variable, **`$FoundVirus`**, but it is local, not global. You can add it to the globally-declared variables and its state will persist across functions called after **`filter_begin()`**.

A caution: until recent versions of MIMEDefang, the **`filter_begin()`** function did not take any parameters. If you find older code examples on the 'Net, be sure to account for the version of MIMEDefang when using them in your own filter file.

`filter`

Called for any MIME part of a message that does not itself contain other MIME parts nested within it, this function is optional. The MIMEDefang documentation refers to these MIME parts as a *leaf part*. It's possible for malware to hide in MIME parts, especially malformed MIME parts, so evaluating these parts is important.

One tip for all functions after **`filter_begin()`** is to use the **`return if message_rejected()`** construct early in the function, as demonstrated in the sample code. This allows functions after **`filter_begin()`** to exit before wasting a lot of time/effort if the E-Mail has already been marked for rejection.

The sample code shows several useful checks, including filtering on “bad” filenames and malformed MIME. Additionally, if you have a global variable to check, you can see what the virus scanner returned for the message, and drop the infected part.

Again, exactly how you respond to these things is up to you. The example code included in the MIMEDefang package may or may not be appropriate to your environment. Also, consider exempting your own hosts, as appropriate, from some or all of the checks. If you wish to filter based on the character set used by the message, this is a good place to do it, but be careful.

`filter_multipart`

Like **`filter()`**, this function is optional. It's called for any MIME part that contains other parts (a *non-leaf part*). The same checks performed in **`filter()`** are generally applicable here, and the remarks above for that function also apply.

`filter_end`

This function marks the final steps of the filtering process, and if the E-Mail has not yet been rejected, then the output of this function will determine its fate. By now, hopefully, you've eliminated most of the “dumb” things that spammers and malware writers do when they spew their garbage.

It's at this point you submit the E-Mail to SpamAssassin for an evaluation. Again, consider

exempting your own hosts from this check, if appropriate. The example code included with MIMEDefang shows the general process of handing the message off to SpamAssassin and evaluating the results. Adding headers and the SpamAssassin report are also shown.

It would also be at this point where you would implement additional features, such as adding to the SPAM score if the message had a virus, or checking some database of per-recipient SPAM limits (where messages with a SPAM score over a certain amount, defined by each recipient, would be rejected or discarded). If you do implement per-recipient SPAM limits, read carefully the documentation on the **`delete_recipient()`** function and how it works. If you preserved the RBL information from an earlier function, you may choose to use it to enhance the SPAM score at this point as well.

If the message had a significant SPAM score, consider stripping any **`X-Spam-Score`** or similar headers from the message, prior to adding your own. Spammers have no compunction about spoofing such headers to fool client-based anti-SPAM tools.

At the end of **`filter_end()`**, the default filter file also references two other functions, **`remove_redundant_html_parts()`** and **`action_rebuild()`**. You should carefully read the documentation for both before implementing them. Consider exempting your own hosts from their effects, if appropriate. E-Mails that contain both text and HTML parts are quite wasteful, but you may not be able to accept the risks of the removal code making a mistake. Performing an **`action_rebuild()`** on all inbound E-Mail is a handy way to ensure that your mail clients receive only E-Mail that conforms to standard MIME, even if the sender is using a broken client to generate the E-Mail. This can help foil some exploits that rely on using oddball MIME constructs to trigger software errors.

If, by now, you haven't rejected, **TEMPFAIL**ed, discarded or otherwise ditched the message, then when this function exits, the E-Mail will move to the next step in delivery.

Summary

Spammers are narcissistic leeches. They can't send their junk from an IP address space they own; it'll be rapidly blacklisted and they'll have to keep changing IPs. So they hijack 'botted Windows PCs with broadband connections and steal the bandwidth and processing power of others to spew their garbage. Then they steal your bandwidth, CPU and disk trying to hammer that garbage through your defenses and fill up your user's mailboxes.

Don't let them.

Use the tools and techniques presented here, and/or from elsewhere around the 'Net, to identify spammers the moment they connect. Reject their obviously fraudulent E-Mail as early as possible in the SMTP conversation: the sooner you do it, the less they get to leech from you.

Reject early, reject often.

Tips for a More Effective Anti-SPAM defense

MIMEDefang

1) Remember dependencies: For example, MIMEDefang supports the "message quarantining" feature added in sendmail v8.13.x. If you have an earlier version of sendmail, you should avoid any sendmail-specific quarantine functions, such as **`action_sm_quarantine()`**. Use of the MIMEDefang Quarantine directory, however, is independent of sendmail version.

2) Be careful with add-on software (*e.g.* **Anomy HTML Cleaner**): There are a number of add-on bits of software for MIMEDefang, or packages that can be integrated with it via sockets (*e.g.* ClamAV) or an API (*e.g.* SpamAssassin). Many of these are fine, but always check their provenance and currency. In particular, the **Anomy HTML Cleaner** program is known to be buggy (and as a consequence, MIMEDefang no longer recommends its use), and the **`File::Scan`** Perl module is likewise suspect. Make sure you're using the right tool, and the right version.

3) Don't be **too** clever: Anyone who's programmed for awhile has probably had the experience of writing some code to perform a task, then going back, looking at it, and saying to themselves "I made this harder than it has to be". Keep your MIMEDefang filter code simple, don't let it become an over-written beast.

For example, consider **`filter_helo()`**, as presented in the sample code above. One might be tempted to add a check for IP **`HELO`** strings to make sure the IP address represented is a valid/routable IP address. However, simply comparing that **`HELO`** string against the actual IP address of the connecting host insures that an invalid/non-routable address won't be accepted as a valid **`HELO`**. The point here is not to over-think the problem.

4) Make sure your hardware is adequate to the task: MIMEDefang and SpamAssassin are both written in Perl, and the MIMEDefang Multiplexor is designed to provide a pool of available slave processes to avoid the CPU cost of starting up a new Perl process every time an E-Mail arrives. The downside is that memory consumption can dramatically increase. At a minimum, the MIMEDefang system configuration proposed in this paper can chew up 100MB of RAM, just for ClamAV and the MIMEDefang processes (*aka* slaves). If your system can't handle it, E-Mail delivery will be delayed or even fail.

5) Understand how MIMEDefang processes are assigned by the Multiplexor: This is discussed at some length in the documentation, but it bears repeating. Once the MIMEDefang process used to filter a given message at a given step is done, it will be made available to the next MILTER call from sendmail, and may be involved in a completely different step of a completely different SMTP conversation. This is important to understand: just because a given MIMEDefang slave was used to apply **`filter_begin()`** through **`filter_end()`** to the last E-Mail received does not insure that the same MIMEDefang slave will be used for those functions when the next E-Mail arrives. Thus, preserving state information between MIMEDefang slave processes is difficult at best, and should be considered an advanced topic (outside the scope of this paper).

SpamAssassin

1) Be prepared to tweak the scoring: SpamAssassin comes with a set of default scoring rules that it uses to grade SPAM. These rules and their scores may or may not be appropriate to your environment. You can modify the scoring for rules by adding your own custom scoring entries to *sa-mimedefang.cf*. See the SpamAssassin documentation for how to write custom scoring rules.

2) Keep it updated: SpamAssassin is a tool under active development. Spammers are also changing tactics in their continuing attempt to flood your mail system with garbage. In the configurations presented by this paper, SpamAssassin is the final line of defense, the trap for the "smart" spammer who has managed to evade detection earlier in the process. Keeping SpamAssassin up to date maintains the integrity of that final defensive line.

3) Pay attention to score reports on SPAM that leaks through: Depending on how you code your filter, you can have SpamAssassin score reports added to E-Mails. These score reports will tell you what tests scored a "hit" and how many points were scored by the test. If you begin to see SPAM leaking through that consistently triggers a particular test (or tests), you can increase the probability of the E-Mail being tagged appropriately by adding a custom scoring rule, as in 1) above. But you have to pay attention to what leaks through. This is a sort of "manual Bayesian learning" - which is not to say that you shouldn't implement SpamAssassin's automatic Bayesian functions (a topic outside the scope of this paper), but this method can sometimes generate better results in the short term.

ClamAV

1) Use **freshclam**: Not directly a part of your anti-SPAM defenses, the **freshclam** component of the ClamAV system is a daemon that checks for updates to the **clamd** virus signature database. Keeping your signatures fresh is an important part of maintaining your anti-virus system.

2) Keep the code updated: The ClamAV system is under active development, and new versions have (historically) been released every 45 to 60 days. Consider subscribing to the ClamAV Announcements mailing list[22] for notifications.

sendmail

1) Upgrading sendmail means rebuilding/upgrading MIMEDefang: MIMEDefang's executables link against sendmail's **libmilter** library. If you upgrade sendmail, you should re-compile/re-link MIMEDefang, or oddball errors may occur[23].

2) Don't forget sendmail's defenses: In my paper **Practical Modern sendmail Configuration**[24], I present a sendmail configuration that includes a number of anti-SPAM features, including using macros such as **`confBAD_RCPT_THROTTLE`**, **`confCONNECTION_RATE_THROTTLE`**, **`confMAX_RCPTS_PER_MESSAGE`**, **`confPRIVACY_FLAGS`**, and a number of **`confTO_*`** (timeout)

macros. The configuration is also MIMEDefang-friendly. These sendmail options are very useful in combating SPAM, and the timeout settings, in particular, help prevent misuse of your mail relay's resources.

Generally

1) Write your start/stop scripts properly: As has been noted, the software configurations presented in this paper create an interdependent system. The **clamd** socket must exist, or MIMEDefang will be unable to start. The MIMEDefang sockets must exist, or sendmail will be unable to start. SpamAssassin operates as part of MIMEDefang, and so receives special dispensation.

Your system start/stop scripts should account for this, both in their timing and operation. **clamd** should not be stopped while MIMEDefang is still running, and sendmail shouldn't start unless MIMEDefang is already running. Consider coding the start/stop scripts for these tools to check for the PID files/numbers of the daemons upon which they are dependent.

2) Pay attention to your logs: Make sure the system is working the way you planned. MIMEDefang, in particular, can generate prodigious logging output, which is very useful for making sure that the filter works the way you intended. You don't want to find out you've been silently discarding every E-Mail sent by the high-profile customer to your corporate CEO after it's been going on for a week.

Helpful Reference Materials

This sample *sa-mimedefang.conf* file shows how to code a MIMEDefang-specific SpamAssassin configuration:

```
#########################################################################
# /path/to/mimedefang/conf/sa-mimedefang.cf
#
# SpamAssassin configuration file for invocation by MIMEDefang
#
# Change Log:
# Who  When        What
# ---- ----------  ----------------------------
#
#########################################################################
# Sample Formats:
#
#   required_hits n
#               (how many hits are required to tag a mail as spam.)
#
#   score SYMBOLIC_TEST_NAME n
#               (if this is omitted, 1 is used as a default score.
#               Set the score to 0 to ignore the test.)
#
# # starts a comment, whitespace is not significant.
#
```

```
# NOTE!  In conjunction with MIMEDefang, SpamAssassin can *NOT* make any
# changes to the message header or body.  Any SpamAssassin settings that
# relate to changing the message will have *NO EFFECT* when used from
# MIMEDefang.  Instead, use the various MIMEDefang Perl functions if you
# need to alter the message.
############################################################################

############################################################################
# Spam score threshold (below this and SpamAssassin will not report that
#     the E-Mail is SPAM; however, a report will still be returned)
required_hits          10

# Whitelist and blacklist addresses are *not* patterns; they're just normal
# strings.  one exception is that "*@isp.com" is allowed.  They should be in
# lower-case.  You can either add multiple addresses on one line,
# whitespace-separated, or you can use multiple lines.
#
#EXAMPLE: whitelist_from           monty@roscom.com

# Add your blacklist entries in the same format...
#EXAMPLE: blacklist_from           friend@public.com
blacklist_from                     *@optonline.com

# Mail using languages used in these country codes will not be marked
# as being possibly spam in a foreign language.
ok_locales                    en

# By default, the subject lines of suspected spam will be tagged.
# This can be disabled here, and should be when SA is invoked via MD
rewrite_subject 0

# By default, spamassassin will include its report in the body
# of suspected spam. Enabling this causes the report to go in the
# headers instead. Using 'use_terse_report' for this is recommended.
report_header 1

# By default, SpamAssassin uses a fairly long report format.
# Enabling this uses a shorter format that includes information
# without lengthy explanations
use_terse_report 0

# By default, spamassassin will change the Content-type: header of
# suspected spam to "text/plain". Since changes made by SA do not
# propagate back through MD, set this to 0.
defang_mime 0

# By default, SpamAssassin will run RBL checks. If you implement
# RBL checks in MD, set this to 1.
skip_rbl_checks 1

# Disable Distributed Checksum
use_dcc 0

# Disable Pryzor
use_pyzor 0
```

```
# Disable Razor2
use_razor2 0

# Define internal hosts as trusted
trusted_networks 127.0.0.1

# Disable DNS support
dns_available no

# Auto-whitelisting database
#auto_whitelist_path /path/to/sa/db

# Bayesian database settings
#bayes_path /path/to/sa/db
#lock_method flock

############################################################################
# Add your own customised scores for some tests below.  The default scores are
# read from the installed "spamassassin.cf" file, but you can override them
# here.  To see the list of tests and their default scores, go to
# http://spamassassin.taint.org/tests.html .
############################################################################

##############################
## End of sa-mimedefang.cf ##
##############################
```

Next, a simple yet effective *mimedefang-filter* file is available via the website associated with this paper[25]. As you look at it, you may note it does not employ some of the more-sophisticated possible functions, such as SPF record checks, greylisting, SpamAssassin Bayesian learning, or a mechanism for per-address SPAM score limits. These omissions are deliberate, as the purpose is to demonstrate, not implement all possible functionality. Also, its functions are deliberate variations on the sample code presented earlier in this paper – the goal being to show different possible approaches, rather than to repeat a single approach.

Finally, the following simple script is helpful in MIMEDefang administration, and allows you to quickly check the "sanity", or syntactical correctness, of your *mimedefang-filter* file. It will not detect logic errors or program design mistakes, it merely makes sure that your filter file is "legal" Perl.

```
#!/path/to/bash
#
# /path/to/mimedefang/conf/sanity.sh
#
# Sanity-check for mimedefang-filter file
#
# Returns 0 if filter is broken, 1 if filter is OK
echo ' '
echo "    Sanity-checking /path/to/mimedefang/conf/mimedefang-filter..."
test123=`/path/to/perl /path/to/mimedefang/bin/mimedefang.pl -test`
echo "    ..$test123"
```

```
chk=`echo $test123 | grep -c "syntactically correct"`
echo ' '
exit $chk
# End of script
```

Footnotes

[1] http://www.crn.com/security/197008347
[2] http://www.informationweek.com/showArticle.jhtml?articleID=197001430
[3] http://www.itwire.com/content/view/15221/53/
[4] Within the scope of this paper, "effective" is defined as having a minimum of false-positives (*e.g.* legitimate E-mail being tagged as SPAM) while stopping the vast majority of actual SPAM. In this context, "effective" doesn't mean "perfect". Some SPAM is expected to leak through, while an occasional "legitimate" message may be erroneously flagged or dropped.
[5] http://www.eff.org/wp/?f=SpamCollateralDamage.html
[6] http://www.mimedefang.org
[7] http://spamassassin.apache.org
[8] http://www.clamav.net
[9] http://www.sendmail.org
[10] http://www.sendmail.org/doc/sendmail-current/libmilter/docs
[11] http://www.sendmail.org/doc/sendmail-current/libmilter/README
[12] http://razor.sourceforge.net
[13] http://www.rhyolite.com/anti-spam/dcc
[14] http://pyzor.sourceforge.net
[15] http://www.milter.org
[16] http://spambayes.sourceforge.net/
[17] Within the MIMEDefang community, there is some contention as to if it is more efficient to run SpamAssassin as its own daemon, using the **spamc/spamd** interface; or as an integrated component of MIMEDefang (as described in the MIMEDefang installation documentation). Materials available through the MIMEDefang website and wiki describe how to implement the alternative (**spamd**) configuration. The topic is considered outside the scope of this paper; however, without getting into the pros and cons, if you already have **spamd** running as a daemon, consider that alternative configuration for your environment.
[18] http://www.postfix.org
[19] As of sendmail v8.14.0, sendmail offers some filtering of **HELO** strings, including the ability to reject some obviously fraudulent strings; however, the MIMEDefang configurations presented in this paper are more flexible, and arguably much more effective, than sendmail's built-in capabilities.
[20] Another approach would be to use a data source, such as LDAP, for either sendmail or MIMEDefang to consult at this step. A perfectly viable idea, it's also outside of the scope of this paper.
[21] In order for such logs to be recorded, you must appropriately configure the **syslogd** daemon.
[22] http://lists.clamav.net/mailman/listinfo/clamav-announce
[23] http://lists.roaringpenguin.com/pipermail/mimedefang/2006-July/030684.html
[24] http://dave.trianglenug.org
[25] http://dave.trianglenug.org/mimedefang-filter.paper

Change Log

Version	Date	Change
0.50	2006-Jul-11	Initial creation
1.00	2006-Aug-12	Published (PDF only)
1.01	2006-Aug-13	Fixed minor typos
1.20	2007-Jan-09	Minor edits; corrected Perl code; updated versions of softwares referenced
1.25	2007-Jan-12	Fixed minor typos; added information about logging
1.26	2007-Mar-20	Created HTML version (not published)
1.30	2007-Mar-23	Added SpamBayes; fixed minor Perl bug; updated versions of software referenced; minor edits; reconciled with HTML version (both published)
1.35	2007-Apr-26	Revised “Philosophy” section; many minor text edits; updated ClamAV version; formatting changes
1.36	2007-May-03	Minor typo and formatting fixes
1.37	2007-May-13	Minor change to “Philosophy” section
1.40	2007-Jun-04	Updated software versions
1.50	2007-Nov-11	Updated software versions; added article citations for SPAM statistics