# Spam Filtering for Mail Exchangers

## How to reject junk mail in incoming SMTP transactions.


### Tor Slettnes

<tor@slett.net>

**Edited by**

### Joost De Cock

### Devdas Bhagat

### Tom Wright



Version 1.0 –– Release Edition

# Table of Contents

# Table of Contents

# Table of Contents

# Introduction

## 1. Purpose of this Document

This document discusses various highly effective and low impact ways to weed out spam and malware during incoming SMTP transactions in a mail exchanger (MX host), with an added emphasis on eliminating so−called *Collateral Spam*.

The discussions are conceptual in nature, but a sample implementation is provided using the Exim MTA and other specific software tools. Miscellaneous other bigotry is expressed throughout.

## 2. Audience

The intended audience is mail system administrators, who are already familiar with such acronyms as SMTP, MTA/MDA/MUA, DNS/rDNS, and MX records. If you are an end user who is looking for a spam filtering solution for your mail reader (such as Evolution, Thunderbird, Mail.app or Outlook Express), this document is *not* for you; but you may wish to point the mail system administrator for your domain (company, school, ISP...) to its existence.

## 3. New versions of this document

The newest version of this document can be found at http://slett.net/spam−filtering−for−mx/. Please check back periodically for corrections and additions.

## 4. Revision History

**Revision History**

| | | |
|---|---|---|
| Revision 1.0 | 2004−09−08 | Revised by: TS |
| First public release. | | |
| Revision 0.18 | 2004−09−07 | Revised by: TS |
| Incorporated second language review from Tom Wright. | | |
| Revision 0.17 | 2004−09−06 | Revised by: TS |
| Incorporated language review from Tom Wright. | | |
| Revision 0.16 | 2004−08−13 | Revised by: TS |
| Incorporated third round of changes from Devdas Bhagat. | | |
| Revision 0.15 | 2004−08−04 | Revised by: TS |
| Incorporated second round of changes from technical review by Devdas Bhagat. | | |
| Revision 0.14 | 2004−08−01 | Revised by: TS |
| Incorporated technical review comments/corrections from Devdas Bhagat. | | |
| Revision 0.13 | 2004−08−01 | Revised by: TS |
| Incorporated technical review from Joost De Cock. | | |
| Revision 0.12 | 2004−07−27 | Revised by: TS |
| Replaced "A Note on Controversies" with a more opinionated "The Good, The Bad, the Ugly" section. Also rewrote text on DNS blocklists. Some corrections from Seymour J. Metz. | | |

Revision 0.11 2004−07−19 Revised by: TS

Incorporated comments from Rick Stewart on RMX++. Swapped order of "Techniques" and "Considerations". Minor typographic fixes in Exim implementation.

Revision 0.10 2004−07−16 Revised by: TS

Added <?dbhtml..?> tags to control generated HTML filenames − should prevent broken links from google etc. Swapped order of "Forwarded Mail" and "User Settings". Correction from Tony Finch on Bayesian filters; commented out check for Subject:, Date:, and Message−ID: headers per Johannes Berg; processing time subtracted from SMTP delays per suggestion from Alan Flavell.

Revision 0.09 2004−07−13 Revised by: TS

Elaborated on problems with envelope sender signatures and mailing list servers, and a scheme to make such signatures optional per host/domain for each user. Moved "Considerations" section out as a separate chapter; added subsections "Blocking Access to other SMTP Server", "User Settings" and "Forwarded Mail". Incorporated Matthew Byng−Maddick's comments on the mechanism used to generate these signatures, Chris Edwards' comments on sender callout verification, and Hadmut Danisch's comments on RMX++ and other topics. Changed license terms (GPL instead of GFDL).

Revision 0.08 2004−07−09 Revised by: TS

Additional work on Exim implementation: Added section on per−user settings and data for SpamAssassin per suggestion from Tollef Fog Heen. Added SPF checks via Exiscan−ACL. Corrections from Sam Michaels.

Revision 0.07 2004−07−08 Revised by: TS

Made corrections to the Exim Envelope Sender Signatures examples, and added support for users to "opt in" to this feature, per suggestion from Christian Balzer.

Revision 0.06 2004−07−08 Revised by: TS

Incorporated Exim/MySQL greylisting implementation and various corrections from Johannes Berg. Moved "Sender Authorization Schemes" up two levels to become a top−level section in the Techniques chapter. Added greylisting for NULL empty envelope senders after DATA. Added SpamAssassin configuration to match Exim examples. Incorporated corrections from Dominik Ruff, Mark Valites, "Andrew" at Supernews.

Revision 0.05 2004−07−07 Revised by: TS

Eliminated the (empty) Sendmail implementation for now, to move ahead with the final review process.

Revision 0.04 2004−07−06 Revised by: TS

Reorganized layout a little: Combined "SMTP−Time Filtering", "Introduction to SMTP", and "Considerations" into a single "Background" chapter. Split the previous "Building ACLs" section in the Exim implementation into top−level sections. Added alternate sender authorization schemes to SPF: Microsoft Caller−ID for E−Mail and RMX++. Incorporated comments from Ken Raeburn.

Revision 0.03 2004−07−02 Revised by: TS

Added discussion on Multiple Incoming Mail Exchangers; minor corrections related to Sender Callout Verification.

Revision 0.02 2004−06−30 Revised by: TS

Added Exim implementation as an appendix

Revision 0.01 2004−06−16 Revised by: TS

Initial draft.

# 5. Credits

A number of people have provided feedback, corrections, and contributions, as indicated in the Revision History. Thank you!

The following are *some* of the people and groups that have provided tools and ideas to this document, in no particular order:

- Evan Harris `<eharris (at) puremagic.com>`, who conceived and wrote a white paper on greylisting.
- Axel Zinser `<fifi (at) hiss.org>`, who apparently conceived of teergrubing.
- The developers of SPF, RMX++, and other Sender Authorization Schemes.
- The creators and maintainers of distributed, collaborative junk mail signature repositories, such as DCC, Razor, and Pyzor.
- The creators and maintainers of various DNS blocklists and whitelists, such as SpamCop, SpamHaus, SORBS, CBL, and many others.
- The developers of SpamAssassin, who have taken giant leaps forward in developing and integrating various spam filtering techniques into a sophisticated heuristics−based tool.
- Tim Jackson `<tim (at) timj.co.uk>` collated and maintains a list of bogus virus warnings for use with SpamAssassin.
- A lot of smart people who developed the excellent Exim MTA, including: Philip Hazel `<ph10 (at) cus.cam.ac.uk>`, the maintainer; Tom Kistner `<tom (at) duncanthrax.net>`, who wrote the Exiscan−ACL patch for SMTP−time content checks; Andreas Metzler `<ametzler (at) debian.org>`, who did a really good job of building the Exim 4 Debian packages.
- Many, many others who contributed ideas, software, and other techniques to counter the spam epidemic.
- You, for reading this document and your interest in reclaiming e−mail as a useful communication tool

# 6. Feedback

I would love to hear of your experiences with the techniques outlined in this document, and of any other comments, questions, suggestions, and/or contributions you may have. Please send me an e−mail at: `<tor@slett.net>`.

If you are able to provide implementations for other *Mail Transport Agent*s, such as Sendmail or Postfix, please let me know.

# 7. Translations

No translations exist yet. If you would like to create one, please let me know.

# 8. Copyright information

# 9. What do you need?

The techniques described in this document predicate system access to the inbound *Mail Exchanger*(s) for the internet domain where you receive e–mail. Essentially, you need to be able to install software and/or modify the configuration files for the *Mail Transport Agent* on that system.

Although the discussions in this document are conceptual in nature and can be incorporated into a number of different MTAs, a sample Exim 4 implementation is provided. This implementation, in turn, incorporates other software tools, such as SpamAssassin. See Appendix A for details.

# 10. Conventions used in this document

The following typographic and usage conventions occur in this text:

**Table 1. Typographic and usage conventions**

| Text type | Meaning |
|---|---|
| "Quoted text" | Quotes from people, quoted computer output. |
| `terminal view` | Literal computer input and output captured from the terminal, usually rendered with a light grey background. |
| **command** | Name of a command that can be entered on the command line. |
| `VARIABLE` | Name of a variable or pointer to content of a variable, as in `$VARNAME`. |
| `option` | Option to a command, as in "the `-a` option to the **ls** command". |
| *`argument`* | Argument to a command, as in "read **man *`ls`*** ". |
| **command `options` *`arguments`*** | Command synopsis or general usage, on a separated line. |
| `filename` | Name of a file or directory, for example "Change to the `/usr/bin` directory." |
| **Key** | Keys to hit on the keyboard, such as "type **Q** to quit". |
| Button | Graphical button to click, like the OK button. |
| Menu–>Choice | Choice to select from a graphical menu, for instance: "Select Help–>About Mozilla in your browser." |
| *Terminology* | Important term or concept: "The Linux *kernel* is the heart of the system." |
| See Glossary | link to related subject within this guide. |
| The author | Clickable link to an external web resource. |

# 11. Organization of this document

This document is organized into the following chapters:

*Background*
: General introduction to SMTP time filtering, and to SMTP.

*Techniques*
  Various ways to block junk mail in an SMTP transaction.

*Considerations*
  Issues that pertain to transaction time filtering.

*Questions & Answers*
  My attempt at anticipating your questions, and then answering them.

A sample Exim implementation is provided in Appendix A.

# Chapter 1. Background

Here we cover the advantages of filtering mail during an incoming SMTP transaction, rather than following the more conventional approach of offloading this task to the mail routing and delivery stage. We also provide a brief introduction to the SMTP transaction.

## 1.1. Why Filter Mail During the SMTP Transaction?

### 1.1.1. Status Quo

If you receive spam, raise your hands. Keep them up.

If you receive computer virii or other malware, raise your hands too.

If you receive bogus *Delivery Status Notification*s (DSNs), such as "Message Undeliverable", "Virus found", "Please confirm delivery", etc, related to messages you never sent, raise your hands as well. This is known as *Collateral Spam*.

This last form is particularly troublesome, because it is harder to weed out than "standard" spam or malware, and because such messages can be quite confusing to recipients who do not possess godly skills in parsing message headers. In the case of virus warnings, this often causes unnecessary concern on the recipient's end; more generally, a common tendency will be to ignore all such messages, thereby missing out on legitimate DSNs.

Finally, I want those of you who have lost legitimate mail into a big black hole – due to misclassification by spam or virus scanners – to lift your feet.

If you were standing before and are still standing, I suggest that you may not be fully aware of what is happening to your mail. If you have been doing any type of spam filtering, even by manually moving mails to the trash can in your mail reader, let alone by experimenting with primitive filtering techniques such as DNS blacklists (SpamHaus, SPEWS, SORBS...), chances are that you have lost some valid mail.

### 1.1.2. The Cause

Spam, just like many other artifacts of greed, is a social disease. Call it affluenza, or whatever you like; lower life forms seek to destroy a larger ecosystem, and if successful, will actually end up ruining their own habitat in the end.

Larger social issues and philosophy aside: You – the mail system administrator – face the very concrete and real life dilemma of finding a way to deal with all this junk.

As it turns out, there are some limitations with the conventional way that mail is being processed and delegated by the various components of mail transport and delivery software. In a traditional setup, one or more *Mail Exchanger*(s) accept most or all incoming mail deliveries to addresses within a domain. Often, they then forward the mail to one or more internal machines for further processing, and/or delivery to the user's mailboxes. If any of these servers discovers that it is unable to perform the requested delivery or function, it generates and returns a DSN back to the sender address in the original mail.

As organizations started deploying spam and virus scanners, they often found that the path of least resistance was to work these into the message delivery path, as mail is transferred from the incoming *Mail Exchanger*(s) to internal delivery hosts and/or software. For instance, a common way filter out spam is by *routing* the mail through SpamAssassin or other software before it is delivered to a user's mailbox, and/or rely on spam filtering capabilities in the user's *Mail User Agent*.

Options for dealing with mail that is classified as spam or virus at this point are limited:

- You can return a *Delivery Status Notification* back to the sender. The problem is that nearly all spam and e−mail borne virii are delivered with faked sender addresses. If you return this mail, it will invariably go to innocent third parties −− perhaps warning a grandmother in Sweden, who uses Mac OS X and does not know much about computers, that she is infected by the Blaster worm. In other words, you will be generating *Collateral Spam*.
- You can drop the message into the bit bucket, without sending any notification back to the sender. This is an even bigger problem in the case of *False Positive*s, because neither the sender nor the receiver will ever know what happened to the message (or in the receiver's case, that it ever existed).
- Depending on how your users access their mail (for instance, if they access it via the IMAP protocol or use a web−based mail reader, but not if they retreive it over POP−3), you may be able to file it into a separate junk folder for them −− perhaps as an option in their account settings.

  This may be the best of these three options. Even so, the messages may remain unseen for some time, or simply overlooked as the receiver more−or−less periodically scans through and deletes mail in their "Junk" folder.

## 1.1.3. The Solution

As you would have guessed by now, the *One True* solution to this problem is to do spam and virus filtering during the SMTP dialogue from the remote host, as the mail is being received by the inbound mail exchanger for your domain. This way, if the mail turns out to be undesirable, you can issue a SMTP *reject* response rather than face the dilemma described above. As a result:

- You will be able to stop the delivery of most junk mail early in the SMTP transaction, before the actual message data has been received, thus saving you both network bandwidth and CPU processing.
- You will be able to deploy some spam filtering techniques that are not possible later, such as SMTP transaction delays and Greylisting.
- You will be able to notify the sender in case of a delivery failure (e.g. due to an invalid recipient address) without directly generating *Collateral Spam*

  We will discuss how you can avoid causing collateral spam indirectly as a result of rejecting mail forwarded from trusted sources, such as mailing list servers or mail accounts on other sites [1].
- You will be able to protect yourself against collateral spam from others (such as bogus "You have a virus" messages from anti−virus software).

OK, you can lower your hands now. If you were standing, and your feet disappeared from under you, you can now also stand up again.

## 1.2. The Good, The Bad, The Ugly

Some filtering techniques are more suitable for use during the SMTP transaction than others. Some are simply better than others. Nearly all have their proponents and opponents.

Needless to say, these controversies extend to the methods described here as well. For instance:

- Some argue that DNS checks penalize individual mail senders purely based on their Internet Service Provider (ISP), not on the merits of their particular message.
- Some point out that ratware traps like SMTP transaction delays and Greylisting are easily overcome and will be less effective over time, while continuing to degrade the Quality of Service for legitimate mail.
- Some find that Sender Authorization Schemes like the Sender Policy Framework give ISPs a way to lock their customers in, and do not adequately address users who roam between different networks or who forward their e−mail from one host to another.

I will steer away from most of these controversies. Instead, I will try to provide a functional description of the various techniques available, including their possible side effects, and then talk a little about my own experiences using some of them.

That said, there are some filtering methods in use today that I deliberately omit from this document:

- Challenge/response systems (like TMDA). These are not suitable for SMTP time filtering, as they rely on first accepting the mail, then returning a confirmation request to the *Envelope Sender*. This technique is therefore outside the scope of this document. [2]
- *Bayesian Filters*. These require training specific to a particular user, and/or a particular language. As such, these too are not normally suitable for use during the SMTP transaction (But see User Settings and Data).
- *Micropayment Schemes* are not really suitable for weeding out junk mail until all the world's legitimate mail is sent with a virtual *postage stamp*. (Though in the mean time, they can be used for the opposite purpose − that is, to accept mail carrying the stamp that would otherwise be rejected).

Generally, I have attempted to offer techniques that are as precise as possible, and to go to great lengths to avoid *False Positive*s. People's e−mail is important to them, and they spend time and effort writing it. In my view, willfully using techniques or tools that reject large amounts of legitimate mail is a show of disrespect, both to the people that are directly affected and to the Internet as a whole. [3] This is especially true for SMTP−time system wide filtering, because end recipients usually have little or no control over the criteria being used to filter their mail.

---

## 1.3. The SMTP Transaction

SMTP is the protocol that is used for mail delivery on the Internet. For a detailed description of the protocol, please refer to RFC 2821, as well as Dave Crocker's introduction to Internet Mail Architecture.

Mail deliveries involve an SMTP transaction between the connecting host (client) and the receiving host (server). For this discussion, the connecting host is the peer, and the receiving host is your server.

In a typical SMTP transaction, the client issues SMTP commands such as **EHLO**, **MAIL FROM:**, **RCPT TO:**, and **DATA**. Your server responds to each command with a 3−digit numeric code indicating whether the command was accepted (**2*xx***), was subject to a temporary failure or restriction (**4*xx***), or failed definitively/permanently (**5*xx***), followed by some human readable explanation. A full description of these codes is included in RFC 2821.

A best case scenario SMTP transaction typically consists of the following relevant steps:

**Table 1−1. Simple SMTP dialogue**

| Client | Server |
|---|---|
| Initiates a TCP connection to server. | Presents an SMTP banner − that is, a greeting that starts with the code **220** to indicate that it is ready to speak SMTP (or usually ESMTP, a superset of SMTP):<br><br>`220 your.f.q.d.n ESTMP...` |
| Introduces itself by way of an Hello command, either **HELO** (now obsolete) or **EHLO**, followed by its own *Fully Qualified Domain Name*:<br><br>`EHLO peers.f.q.d.n` | Accepts this greeting with a **250** response. If the client used the *extended* version of the Hello command (**EHLO**), your server knows that it is capable of handling multi−line responses, and so will normally send back several lines indicating the capabilities offered by your server:<br><br>`250−your.f.q.d.n Hello ...`<br>`250−SIZE 52428800`<br>`250−8BITMIME`<br>`250−PIPELINING`<br>`250−STARTTLS`<br>`250−AUTH`<br>`250 HELP`<br><br>If the **PIPELINING** capability is included in this response, the client can from this point forward issue several commands at once, without waiting for the response to each one. |
| Starts a new mail transaction by specifying the *Envelope Sender*:<br><br>`MAIL FROM:<sender@address>` | Issues a **250** response to indicate that the sender is accepted. |
| Lists the *Envelope Recipient*s of the message, one at a time, using the command:<br><br>`RCPT TO:<receiver@address>` | Issues a response to each command (**2*xx***, **4*xx***, or **5*xx***, depending on whether delivery to this recipient was accepted, subject to a temporary failure, or rejected). |
| Issues a **DATA** command to indicate that it is ready to send the message. | Responds **354** to indicate that the command has been provisionally accepted. |

| | |
|---|---|
| Transmits the message, starting with RFC 2822 compliant header lines (such as: `From:`, `To:`, `Subject:`, `Date:`, `Message-ID:`). The header and the body are separated by an empty line. To indicate the end of the message, the client sends a single period (".") on a separate line. | Replies **250** to indicate that the message has been accepted. |
| If there are more messages to be delivered, issues the next **MAIL FROM:** command. Otherwise, it says **QUIT**, or in rare cases, simply disconnects. | Disconnects. |

# Chapter 2. Techniques

In this chapter, we look at various ways to weed out junk mail during the SMTP transaction from remote hosts. We will also try to anticipate some of the side effects from deploying these techniques.

## 2.1. SMTP Transaction Delays

As it turns out, one of the more effective ways of stopping spam is by imposing transaction delays during an inbound SMTP dialogue. This is a primitive form of *teergrubing*, see: http://www.iks−jena.de/mitarb/lutz/usenet/teergrube.en.html

Most spam and nearly all e−mail borne virii are delivered directly to your server by way of specialized SMTP client software, optimized for sending out large amounts of mail in a very short time. Such clients are commonly known as *Ratware*.

In order to accomplish this task, ratware authors commonly take a few shortcuts that, ahem, "diverge" a bit from the RFC 2821 specification. One of the intrinsic traits of ratware is that it is notoriously impatient, especially with slow−responding mail servers. They may issue the **HELO** or **EHLO** command before the server has presented the initial SMTP banner, and/or try to pipeline several SMTP commands before the server has advertised the **PIPELINING** capability.

Certain *Mail Transport Agent*s (such as Exim) automatically treat such SMTP protocol violations as *synchronization errors*, and immediately drop the incoming connection. If you happen to be using such an MTA, you may already see a lot of entries to this effect in your log files. In fact, chances are that if you perform any time−consuming checks (such as DNS checks) prior to presenting the initial SMTP banner, such errors will occur frequently, as ratware clients simply do not take the time to wait for your server to come alive (Things to do, people to spam).

We can help along by imposing additional delays. For instance, you may decide to wait:

- 20 seconds before presenting the initial SMTP banner,
- 20 seconds after the Hello (**EHLO** or **HELO**) greeting,
- 20 seconds, after the **MAIL FROM:** command, and
- 20 seconds after each **RCPT TO:** command.

Where did 20 seconds come from, you ask. Why not a minute? Or several minutes? After all, RFC 2821 mandates that the sending host (client) should wait up to several minutes for every SMTP response. The issue is that some receiving hosts, particularly those that use Exim, may perform Sender Callout Verification in response to incoming mail delivery attempts. If you or one of your users send mail to such a host, it will contact the *Mail Exchanger* (MX host) for your domain and start an SMTP dialogue in order to validate the sender address. The default timeout of such Sender Callout Verifications is 30 seconds − if you impose delays this long, the peer's sender callout verification would fail, and in turn the original mail delivery from you/your user might be rejected (usually with a temporary failure, which means the message delivery will be retried for 5 days or so before the mail is finally returned to the sender).

In other words, 20 seconds is about as long as you can stall before you start interfering with legitimate mail deliveries.

If you do not like imposing such delays on every SMTP transaction (say, you have a very busy site and are low on machine resources), you may choose to use "selective" transaction delays. In this case, you could impose the delay:

- If there is a problem with the peer's DNS information (see DNS checks).
- After detecting some sign of trouble during the SMTP transaction (see SMTP checks).
- Only in the highest−numbered MX host in your DNS zone, i.e. the mail exchanger with the last priority. Often, *Ratware* specifically target these hosts, whereas legitimate MTAs will try the lower−numbered MX hosts first.

In fact, selective transaction delays may be a good way to incorporate some less conclusive checks that we will discuss in the following sections. You probably do not wish to reject the mail outright based the results from e.g. the SPEWS blacklist, but on the other hand, it may provide a strong enough indication of trouble that you can at least impose transaction delays. After all, legitimate mail deliveries are not affected, other than being subjected to a slight delay.

Conversely, if you find conclusive evidence of spamming (e.g. by way of certain SMTP checks), and your server can afford it, you may choose to impose an extended delay, e.g. 15 minutes or so, before finally rejecting the delivery [4]. This is for little or no benefit other than slowing down the spammer a little bit in their quest to reach as many people as possible before DNS blacklists and other collaborative network checks catch up. In other words, pure altruism on your side. :−)

In my own case, selective transaction delays and the resulting SMTP synchronization errors account for nearly 50% of rejected incoming delivery attempts. This roughly translates into saying that nearly 50% of incoming junk mail is stopped by SMTP transaction delays alone.

See also *What happens when spammers adapt...*.

## 2.2. DNS Checks

Some indication of the integrity of a particular peer can be gleaned directly from the *Domain Name System* (DNS), even before SMTP commands are issued. In particular, various DNS blacklists can be consulted to find out if a particular IP address is known to violate or fulfill certain criteria, and a simple pair of forward/reverse (DNS/rDNS) lookups can be used as a vague indicator of the host's general integrity.

Moreover, various data items presented during the SMTP dialogue (such as the name presented in the Hello greeting) can be subjected to DNS validation, once it becomes available. For a discussion on these items, see the section on SMTP checks, below.

A word of caution, though. DNS checks are not always conclusive (e.g. a required DNS server may not be responding), and not always indicative of spam. Moreover, if you have a very busy site, they can be expensive in terms of processing time per message. That said, they can provide useful information for logging purposes, and/or as part of a more holistic integrity check.

### 2.2.1. DNS Blacklists

DNS blacklists (DNSbl's, formerly called "Real−time Black−hole Lists" after the original blacklist, "mail−abuse.org") make up perhaps the most common tool to perform transaction−time spam blocking. The receiving server performs one or more rDNS lookups of the peer's IP address within various DNSbl zones, such as "dnsbl.sorbs.net", "opm.blitzed.org", "lists.dsbl.org", and so forth. If a matching DNS record is found, a typical action is to reject the mail delivery. [5]

If in addition to the DNS address ("A" record) you look up the "TXT" record of an entry, you will typically receive a one−line description of the listing, suitable for inclusion in a SMTP reject response. To try this out, you can use the "host" command provided on most Linux and UNIX systems:

```
host −t txt 2.0.0.127.dnsbl.sorbs.net
```

There are currently hundreds of these lists available, each with different listing criteria, and with different listing/unlisting policies. Some lists even combine several listing criteria into the same DNSbl, and issue different data in response to the rDNS lookup, depending on which criterion affects the address provided. For instance, a rDNS lookup against `sbl−xbl.spamhaus.org` returns 127.0.0.2 for IP addresses that are believed by the SpamHaus staff to directly belong to spammers and their providers, 127.0.0.4 response for *Zombie Host*s, or a 127.0.0.6 response for *Open Proxy* servers.

Unfortunately, many of these lists contain large blocks of IP addresses that are not directly responsible for the alleged violations, don't have clear listing / delisting policies, and/or post misleading information about which addresses are listed[6]. The blind trust in such lists often cause a large amount of what is referred to as *Collateral Damage* (not to be confused with *Collateral Spam*).

For that reason, rather than rejecting mail deliveries outright based on a single positive response from DNS blacklists, many administrators prefer to use these lists in a more nuanced fashion. They may consult several lists, and assign a "score" to each positive response. If the total score for a given IP address reaches a given threshold, deliveries from that address are rejected. This is how DNS blacklists are used by filtering software such as SpamAssassin (Spam Scanners).

One could also use such lists as one of several triggers for SMTP transaction delays on incoming connections (a.k.a. "teergrubing"). If a host is listed in a DNSbl, your server would delay its response to every SMTP command issued by the peer for, say, 20 seconds. Several other criteria can be used as triggers for such delays; see the section on SMTP transaction delays.

### 2.2.2. DNS Integrity Check

Another way to use DNS is to perform a reverse lookup of the peer's IP address, then a forward lookup of the resulting name. If the original IP address is included in the result, its DNS integrity has been validated. Otherwise, the DNS information for the connecting host is not valid.

Rejecting mails based on this criterion may be an option if you are a militant member of the DNS police, setting up an incoming MX for your own personal domain, and don't mind rejecting legitimate mail as a way to impress upon the sender that they need to ask their own system administrator to clean up their DNS records. For everyone else, the result of a DNS integrity check should probably only be used as one data point in a larger set of heuristics. Alternatively, as above, using SMTP transaction delays for misconfigured hosts may not be a bad idea.

## 2.3. SMTP checks

Once the SMTP dialogue is underway, you can perform various checks on the commands and arguments presented by the remote host. For instance, you will want to ensure that the name presented in the Hello greeting is valid.

However, even if you decide to reject the delivery attempt early in the SMTP transaction, you may not want to perform the actual rejection right away. Instead, you may stall the sender with SMTP transaction delays until after the **RCPT TO:**, then reject the mail at that point.

The reason is that some ratware does not understand rejections early in the SMTP transaction; they keep trying. On the other hand, most of them give up if the **RCPT TO:** fails.

Besides, this gives a nice opportunity to do a little *teergrubing*.

## 2.3.1. Hello (HELO/EHLO) checks

Per RFC 2821, the first SMTP command issued by the client should be EHLO (or if unsupported, HELO), followed by its primary, *Fully Qualified Domain Name*. This is known as the Hello greeting. If no meaningful FQDN is available, the client can supply its IP address enclosed in square brackets: "[1.2.3.4]". This last form is known as an IPv4 address "literal" notation.

Quite understandably, *Ratware* rarely present their own FQDN in the Hello greeting. Rather, greetings from ratware usually attempt to conceal the sending host's identity, and/or to generate confusing and/or misleading "Received:" trails in the message header. Some examples of such greetings are:

- Unqualified names (i.e. names without a period), such as the "local part" (username) of the recipient address.
- A plain IP address (i.e. not an IP literal); usually yours, but can be a random one.
- Your domain name, or the FQDN of your server.
- Third party domain names, such as `yahoo.com` and `hotmail.com`.
- Non−existing domain names, or domain names with non−existing name servers.
- No greeting at all.

### 2.3.1.1. Simple HELO/EHLO syntax checks

Some of these RFC 2821 violations are both easy to check against, and clear indications that the sending host is running some form of *Ratware*. You can reject such greetings −− either right away, or e.g. after the **RCPT TO:** command.

First, feel free to reject plain IP addresses in the Hello greeting. Even if you wish to generously allow everything RFC 2821 mandates, recommends, and suggests, you will note that IP addresses should always be enclosed in square brackets when presented in lieu of a name. [7]

In particular, you may wish to issue a strongly worded rejection message to hosts that introduce themselves using *your* IP address − or for that matter, your host name. They are plainly lying. Perhaps you want to stall the sender with an exceedingly long SMTP transaction delay in response to such a greeting; say, hours.

For that matter, my own experience indicates that *no* legitimate sites on the internet present themselves to other internet sites using an IP address literal (the [x.y.z.w] notation) either. Nor should they; all hosts sending mail directly on the internet should use their valid *Fully Qualified Domain Name*. The only use of use of IP literals I have come across is from mail user agents on my local area network, such as Ximian Evolution, configured to use my server as outgoing SMTP server (smarthost). Indeed, I only accept literals from my own LAN.

You may or may not also wish to reject unqualified host names (host names without a period). I find that these are rarely (but not never – how's that for double negative negations) legitimate.

Similarly, you can reject host names that contain invalid characters. For internet domains, only alphanumeric letters and hyphen are valid characters; a hyphen is not allowed as the first character. (You may also want to consider the underscore a valid character, because it is quite common to see this from misconfigured, but ultimately well–meaning, Windows clients).

Finally, if you receive a **MAIL FROM:** command without first having received a Hello greeting, well, polite people greet first.

On my servers, I reject greetings that fail any of these syntax checks. However, the rejection does not actually take place until after the **RCPT TO:** command. In the mean time, I impose a 20 second transaction delay after each SMTP command (**HELO/EHLO**, **MAIL FROM:**, **RCPT TO:**).

#### 2.3.1.2. Verifying the Hello greeting via DNS

Hosts that make it this far have presented at least a superficially credible greeting. Now it is time to verify the provided name via DNS. You can:

- Perform a forward lookup of the provided name, and match the result against the peer's IP address
- Perform a reverse lookup of the peer's IP address, and match it against name provided in the greeting.

If either of these two checks succeeds, the name has been verified.

Your MTA may have a built–in option to perform this check. For instance, in Exim (see Appendix A), you want to set "helo_try_verify_hosts = *", and create ACLs that take action based on the "verify = helo" condition.

This check is a little more expensive in terms of processing time and network resources than the simple syntax checks. Moreover, unlike the syntax checks, a mismatch does not always indicate ratware; several large internet sites, such as hotmail.com, yahoo.com, and amazon.com, frequently present unverifiable Hello greetings.

On my servers, I do a DNS validation of the Hello greeting if I am not already stalling the sender with transaction delays based on prior checks. Then, if this check fails, I impose a 20 second delay on every SMTP command from this point forward. I also prepare a "X–HELO–Warning:" header that I will later add to the message(s), and use to increase the SpamAssassin score for possible rejection after the message data has been received.

### 2.3.2. Sender Address Checks

After the client has presented the **MAIL FROM:** <*`address`*> command, you can validate the supplied *Envelope Sender* address as follows. [8]

#### 2.3.2.1. Sender Address Syntax Check

Does the supplied address conform to the format <*`localpart@domain`*>? Is the *`domain`* part a syntactically valid *Fully Qualified Domain Name*?

Often, your MTA performs these checks by default.

#### 2.3.2.2. Impostor Check

In the case where you and your users send all your outgoing mail only through a select few servers, you can reject messages from other hosts in which the "domain" of the sender address is your own.

A more general alternative to this check is Sender Policy Framework.

#### 2.3.2.3. Simple Sender Address Validation

If the address is local, is the "local part" (the part before the @ sign) a valid mailbox on your system?

If the address is remote, does the "domain" (the part after the @ sign) exist?

#### 2.3.2.4. Sender Callout Verification

This is a mechanism that is offered by some MTAs, such as Exim and Postfix, to validate the "local part" of a remote sender address. In Postfix terminology, it is called "Sender Address Verification".

Your server contacts the MX for the `domain` provided in the sender address, attempting to initiate a secondary SMTP transaction as if delivering mail to this address. It does not actually send any mail; rather, once the **RCPT TO:** command has been either accepted or rejected by the remote host, your server sends **QUIT**.

By default, Exim uses an empty envelope sender address for such callout verifications. The goal is to determine if a *Delivery Status Notification* would be accepted if returned to the sender.

Postfix, on the other hand, defaults to the sender address `<postmaster@domain>` for address verification purposes (`domain` is taken from the `$myorigin` variable). For this reason, you may wish to treat this sender address the same way that you treat the NULL envelope sender (for instance, avoid SMTP transaction delays or Greylisting, but require Envelope Sender Signatures in recipient addresses). More on this in the implementation appendices.

You may find that this check alone may not be suitable as a trigger to reject incoming mail. Occasionally, legitimate mail, such as a recurring billing statement, is sent out from automated services with an invalid return address. Also, an unfortunate side effect of spam is that some users tend to mangle the return address in their outgoing mails (though this may affect the "From:" header in the message itself more often than the *Envelope Sender*).

Moreover, this check only verifies that an address is valid, not that it was authentic as the sender of this particular message (but see also Envelope Sender Signature).

Finally, there are reports of sites, such as "aol.com", that will unconditionally blacklist any system from which they discover sender callout requests. These sites may be frequent victims of *Joe Job*s, and as a result, receive storms of sender callout requests. By taking part in these DDoS (Distributed Denial−of−Servcie) attacks, you are effectively turning yourself into a pawn in the hands of the spammer.

## 2.3.3. Recipient Address Checks

This should be simple, you say. A recipient address is either valid, in which case the mail is delivered, or invalid, in which case your MTA takes care of the rejection by default.

Let us have a look, shall we?

### 2.3.3.1. Open Relay Prevention

*Do not relay mail from remote hosts to remote addresses!* (Unless the sender is authenticated).

This may seem obvious to most of us, but apparently this is a frequently overlooked consideration. Also, not everyone may have a full grasp of the various internet standards related to e–mail addresses and delivery paths (consider "percent hack domains", "bang (!) paths", etc).

If you are unsure whether your MTA acts as an an *Open Relay*, you can test it via "relay–test.mail–abuse.org". At a shell prompt on your server, type:

```
telnet relay-test.mail-abuse.org
```

This is a service that will use various tests to see whether your SMTP server appears to forward mail to remote e–mail addresses, and/or any number of address "hacks" such as the ones mentioned above.

Preventing your servers from acting as open relays is extremely important. If your server is an open relay, and spammers find you, you will be listed in numerous DNS blacklists instantly. If the maintainers of certain other DNS blacklists find you (by probing, and/or by acting on complaints), you will be listed in those for an extended period of time.

### 2.3.3.2. Recipient Address Lookups

This, too may seem banal to most of us. It is not always so.

If your users' mail accounts and mailboxes are stored directly on your incoming mail exchanger, you can simply check that the "local part" of the recipient address corresponds to a valid mailbox. No problem here.

There are two scenarios where verification of the recipient address is more cumbersome:

- If your machine is a backup MX for the recipient domain.
- If your machine forwards all mail for your domain to another (presumably internal) server.

The alternative to recipient address verification is to accept all recipient addresses within these respective domains, which in turn means that you or the destination server might have to generate a *Delivery Status Notification* for recipient addresses that later turn out to be invalid. Ultimately, this means that you would be generating collateral spam.

With that in mind, let us see how we can verify the recipient in the scenarios listed above.

##### 2.3.3.2.1. Recipient Callout Verification

This is a mechanism that is offered by some MTAs, such as Exim and Postfix, to verify the "local part" of a remote recipient address (see *Sender Callout Verification* for a description of how this works). In Postfix terminology, this is called "Recipient Address Verification".

In this case, server attempts to contact the final destination host to validate each recipient address before you, in turn, accept the **RCPT TO:** command from your peer.

This solution is simple and elegant. It works with any MTA that might be running on the final destination host, and without access to any particular directory service. Moreover, if that MTA happens to perform a fuzzy match on the recipient address (this is the case with Lotus Domino servers), this check will accurately reflect whether the recipient address is eventually going to be accepted or not – something which may not be true for the mechanisms described below.

Be sure to keep the original *Envelope Sender* intact for the recipient callout, or the response from the destination host may not be accurate. For instance, it may reject bounces (i.e. mail with no envelope sender) for system users and aliases, as described in Accept Bounces Only for Real Users.

Among major MTAs, Exim and Postfix support this mechanism.

##### 2.3.3.2.2. Directory Services

Another good solution would be a directory service (e.g. one or more LDAP servers) that can be queried by your MTA. The most common MTAs all support LDAP, NIS, and/or various other backends that are commonly used to provide user account information.

The main sticking point is that unless the final destination host of the e−mail already uses such a directory service to map user names to mailboxes, there may be some work involved in setting this up.

##### 2.3.3.2.3. Replicated Mailbox Lists

If none of the options above are viable, you could fall back to a "poor man's directory service", where you would periodically copy a current list of mailboxes from the machine where they are located, to your MX host(s). Your MTA would then consult this list to validate **RCPT TO:** commands in incoming mail.

If the machine(s) that host(s) your mailboxes is/are running on some flavor of UNIX or Linux, you could write a script to first generate such a list, perhaps from the local "/etc/passwd" file, and then copy it to your MX host(s) using the "scp" command from the OpenSSH suite. You could then set up a "cron" job (type **man cron** for details) to periodically run this script.

#### 2.3.3.3. Dictionary Attack Prevention

*Dictionary Attack* is a term used to describe SMTP transactions where the sending host keeps issuing **RCPT TO:** commands to probe for possible recipient addresses based on common names (often alphabetically starting with "aaron", but sometimes starting later in the alphabet, and/or at random). If a particular address is accepted by your server, that address is added into the spammer's arsenal.

Some sites, particularly larger ones, find that they are frequent targets of such attacks. From the spammer's perspective, chances of finding a given username on a large site is better than on sites with only a few users.

One effective way to combat dictionary attacks is to issue increasing transaction delays for each failed address. For instance, the first non−existing recipient address can be rejected with a 20−second delay, the second address with a 30−second delay, and so on.

#### 2.3.3.4. Accept only one recipient for DSNs

Legitimate *Delivery Status Notification*s should be sent to only one recipient address − the originator of the original message that triggered the notification. You can drop the connection if the *Envelope Sender* address is empty, but there are more than one recipients.

## 2.4. Greylisting

The *greylisting* concept is presented by Evan Harris in a whitepaper at: http://projects.puremagic.com/greylisting/.

### 2.4.1. How it works

Like SMTP transaction delays, greylisting is a simple but highly effective mechanism to weed out messages that are being delivered via *Ratware*. The idea is to establish whether a prior relationship exists between the sender and the receiver of a message. For most legitimate mail it does, and the delivery proceeds normally.

On the other hand, if no prior relationship exists, the delivery is temporariliy rejected (with a **451** SMTP response). Legitimate MTAs will treat this response accordingly, and retry the delivery in a little while[9]. In contrast, ratware will either make repeated delivery attempts right away, and/or simply give up and move on to the next target in its address list.

Three pieces of information from a delivery attempt, referred to a as a *triplet* are used to uniquely identify the relationship between a sender and a receiver:

- The *Envelope Sender*.
- The sending host's IP address.
- The *Envelope Recipient*.

If a delivery attempt was temporarily rejected, this triplet is cached. It remains greylisted for a given amount of time (nominally 1 hour), after which it is whitelisted, and new delivery attempts would succeed. If no new delivery attempts occur prior to a given timeout (nominally 4 hours), then the triplet expires from the cache.

If a whitelisted triplet has not been seen for an extended duration (at minimum one month, to account for monthly billing statements and the like), it is expired. This prevents unlimited growth of the list.

These timeouts are taken from Evan Harris' original greylisting whitepaper (or should we say, ahem, "greypaper"?) Some people have found that a larger timeout may be needed before greylisted triplets expire, because certain ISPs (such as *earthlink.net*) retry deliveries only every 6 hours or similar. [10]

### 2.4.2. Greylisting in Multiple Mail Exchangers

If you operate more than one incoming mail exchangers, and each exchanger maintains its own greylisting cache, then:

- First−time deliveries from a given sender to one of your users may theoretically be delayed up to *N* times the initial 1−hour delay, where *N* is the number of mail exchangers. This is because the message would likely be retried at a different server than the one that issued the **451** response to the initial delivery. In the worst case, the sender host may not get around to retrying the delivery to the first exchanger for 4 hours, or until after the greylist triplet has expired, thereby causing the delivery attempt to be rejected over and over again, until the sender gives up (usually after 4 days or so).

  In practice, this is unlikely. If a delivery attempt temporarily fails, the sender host normally retries the delivery immediately, using a different MX. Thus, after one hour, any of these MX hosts would accept the message.
- Even after a triplet has been whitelisted in one of your MXs, the next message with the same triplet will be greylisted if it is delivered to a different MX.

For these reasons, you may want to implement a solution where the database of greylist triplets is shared between your incoming mail exchangers. However, since the machine that hosts this database would become a single point of failure, you would have to take a sensible action if that machine is down (e.g. accept all deliveries). Or you could use database replication techniques and have the SMTP server fall back to one of the replicating servers for lookups.

### 2.4.3. Results

In my own experience, *greylisting* gets rid of about 90% of unique junk mail deliveries, *after* most of the SMTP checks previously described are applied! If you used greylisting as a first defense, it would likely catch an even higher percentage of incoming junk mail.

Conversely, there are virtually zero *False Positive*s resulting from this technique. All major *Mail Transport Agent*s perform delivery retries after a temporary failure, in a manner that will eventually result in a successful delivery.

The downside to greylisting is a legitimate mail from people who have not e−mailed a particular recipient in the past is subject to a one−hour delay (or maybe several hours, if you operate several MX hosts).

See also *What happens when spammers adapt...*.

## 2.5. Sender Authorization Schemes

Various schemes have been developed for sender verification where not only the validity, but also the authenticity, of the sender address is checked. The owner of a internet domain specifies certain criteria that must be fulfilled in authentic deliveries from senders within that domain.

Two early proposed schemes of this kind were:

- `MAIL−FROM` MX records, conceived by Paul Vixie `<paul (at) vix.com>`
- Reverse Mail Exchanger (RMX) records as an addition to DNS itself, conceived and published by Hadmut Danisch `<hadmut (at) danisch.de>`.

Under both of these schemes, all mails from `<user@domain.com>` had to come from the hosts specified in `<domain.com>`'s DNS zone.

These schemes have evolved. Alas, they have also forked.

## 2.5.1. Sender Policy Framework (SPF)

"Server Policy Framework" (previously "Sender Permitted From") is perhaps the most well−known scheme for sender authorization. It is loosely based on the original schemes described above, but allows for a bit more flexibility in the criteria that can be posted by the domain holder.

SPF information is published as a `TXT` record in a domain's top−level DNS zone. This record can specify:

- which hosts are allowed to send mail from that domain
- the mandatory presence of a GPG (GNU Privacy Guard) signature in outgoing mail from the domain
- other criteria; see http://spf.pobox.com/ for details.

The structure of the **TXT** record is still undergoing development, however basic features to accomplish the above are in place. It starts with the string `v=spf1`, followed by such modifiers as:

- `a` − the IP address of the domain itself is a valid sender host
- `mx` − the incoming mail exchanger for that domain is also a valid sender
- `ptr` − if a rDNS lookup of the sending host's IP address yields a name within the domain portion of the sender address, it is a valid sender.

Each of these modifiers may be prefixed with a plus sign (+), minus sign (−), question mark (?), or tilde (~) to indicate whether it specifies an authorative source, an non−authorative source, a neutral stance, or a likely non−authorative source, respectively.

Each modifier may also be extended with a colon, followed by an alternate domain name. For instance, if you are a Comcast subscriber, your own DNS zone may include the string "`−ptr:client.comcast.net ptr:comcast.net`" to indicate that your outgoing e−mail never comes from a host that resolves to *`anything`*.client.comcast.net, but could come from other hosts that resolve to *`anything`*`.comcast.net`.

SPF information is currently published for a number of high−profile internet domains, such as aol.com, altavista.com, dyndns.org, earthlink.net, and google.com.

Sender authorization schemes in general and SPF in particular are not universally accepted. In particular, one objection is that domain holders may effectively establish a monopoly on relaying outgoing mail from their users/customers.

Another objection is that SPF breaks traditional e−mail forwarding − the forwarding host may not have the authority to do so per the SPF information in the envelope sender domain. This is partly addressed via SRS, or *Sender Rewriting Scheme*, wherein the forwarder of the mail will modify the *Envelope Sender* address to the format:

```
user=source.domain@forwarder.domain
```

## 2.5.2. Microsoft Caller−ID for E−Mail

Similar to SPF, in that acceptance criteria are posted via a TXT record in the sending domain's DNS zone. However, rather than relying on simple keywords, MS CIDE information consists of fairly large structures encoded in XML. The XML schema is published under a license by Microsoft.

While SPF would nominally be used to check the *Envelope Sender* address of an e−mail, MS CIDE is mainly a tool to validate the RFC 2822 header of the message itself. Thus, the earliest point at which such a check could be applied would be after the message data has been delivered, before issuing the final **250** response.

Quite frankly, dead on arrival. Encumbered by patent issues and sheer complexity.

That said, Recent SPF tools posted on http://spf.pobox.com/ are capable of checking MS Caller−ID information in addition to SPF.

### 2.5.3. RMX++

(part of *Simple Caller Authorization Framework − SCAF*). This scheme is developed by Hadmut Danisch, who also conceived of the original RMX.

RMX++ allows for dynamic authorization by way of HTTP servers. The domain owner publishes a server location via DNS, and the receiving host contacts that server in order to obtain an *authorization record* to verify the authenticity of the caller.

This scheme allows the domain owner more fine−grained control of criteria used to authenticate the sender address, without having to publicly reveal the structure of their network (as with SPF information in static TXT records). For instance, an example from Hadmut is an authorization server that allows no more than five messages from a given address per day after business hours, then issues an alert once the limit has been reached.

Moreover, SCAF is not limited to e−mail, but can also be used to provide caller authentication for other services such as Voice over IP (VoIP).

One possible downside with RMX++, as noted by Rick Stewart `<rick.stewart (at) theinternetco.net>`, is its impact on machine and network resources: Replies from HTTP servers are not as widely cached as information obtained directly via DNS, and it is signifcantly more expensive to make an HTTP request than a DNS request.

Further, Rick notes that the dynamic nature of RMX++ makes faults harder to track. If there is a five−message−per−day limit, as in the example above, and one message gets checked five times, then the limit is hit with a single message. It makes re−checking a message impossible.

For more information on RMX, RMX++, and SCAF, refer to: http://www.danisch.de/work/security/antispam.html.

## 2.6. Message data checks

Time has come to look at the content of the message itself. This is what conventional spam and virus scanners do, as they normally operate on the message after it has been accepted. However, in our case, we perform these checks *before* issuing the final **250** response, so that we have a chance to reject the mail on the spot rather than later generating *Collateral Spam*.

If your incoming mail exchangers are very busy (i.e. large site, few machines), you may find that performing some or all of these checks directly in the mail exchanger is too costly. In particular, running Virus Scanners and Spam Scanners do take up a fair amount of CPU bandwidth and time.

If so, you will want to set up dedicated machines for these scanning operations. Most server−side anti−spam and anti−virus software can be invoked over the network, i.e. from your mail exchanger. More on this in the following chapters, where we discuss implementation for the various MTAs.

## 2.6.1. Header checks

### 2.6.1.1. Missing Header Lines

RFC 2822 mandates that a message *should* contain at least the following header lines:

```
From: ...
To: ...
Subject: ...
Message-ID: ...
Date: ...
```

The absence of any of these lines means that the message is not generated by a mainstream *Mail User Agent*, and that it is probably junk [11].

### 2.6.1.2. Header Address Syntax Check

Addresses presented in the message header (i.e. the **To:**, **Cc:**, **From:** ... fields) should be syntactically valid. Enough said.

### 2.6.1.3. Simple Header Address Validation

For each address in the message header:

- If the address is local, is the *local part* (before the @ sign) a valid mailbox?
- If the address is remote, does the *domain part* (after the @ sign) exist?

### 2.6.1.4. Header Address Callout Verification

This works similar to Sender Callout Verification and Recipient Callout Verification. Each remote header address is verified by calling the primary MX for the corresponding domain to determine if a *Delivery Status Notification* would be accepted.

## 2.6.2. Junk Mail Signature Repositories

One trait of junk mail is that it is sent to a large number of addresses. If 50 other recipients have already flagged a particular message as spam, why couldn't you use this fact to decide whether or not to accept the message when it is delivered to you? Better yet, why not set up *Spam Traps* that feed a public pool of known spam?

I am glad you asked. As it turns out, such pools do exist:

- Razor
- Pyzor
- Distributed Checksum Clearinghouse (DCC)

These tools have progressed beyond simple signature checks that only trigger if you receive an identical copy of a message that is known to be junk mail. Rather, they evaluate common patterns, to account for slight variations in the message header and body.

### 2.6.3. Binary garbage checks

Messages containing non−printable characters are rare. When they do show up, the message is nearly always a virus, or in some cases spam written in a non−western language, without the appropriate MIME encoding.

One particular case is where the message contains NUL characters (ordinal zero). Even if you decide that figuring out what a *non−printable* character means is more complex than beneficial, you might consider checking for this character. That is because some *Mail Delivery Agent*s, such as the Cyrus Mail Suite, will ultimately reject mails that contain it. [12]. If you use such software, you should definitely consider getting rid of NUL characters.

On the other hand, the (now obsolete) RFC 822 specification did not explicitly prohibit NUL characters in the message. For this reason, as an alternative to rejecting mails containing it, you may choose to strip these characters from the message before delivering it to Cyrus.

### 2.6.4. MIME checks

Similarly, it might be worthwhile to validate the MIME structure of incoming message. MIME decoding errors or inconsistencies do not happen very often; but when they do, the message is definitely junk. Moreover, such errors may indicate potential problems in subsequent checks, such as File Attachment Checks, Virus Scanners, or Spam Scanners.

In other words, if the MIME encoding is illegal, reject the message.

### 2.6.5. File Attachment Check

When was the last time someone sent you a Windows screensaver (".scr" file) or Windows Program Information File (".pif") that you actually wanted?

Consider blocking messages with "Windows executable" file attachment(s) − i.e. file names that end with a period followed by any of a number of three−letter combinations such as the above. This check consumes significantly less resources on your server than Virus Scanners, and may also catch new virii for which a signature does not yet exist in your anti−virus scanner.

For a more−or−less comprehensive list of such "file name extensions", please visit: http://support.microsoft.com/default.aspx?scid=kb;EN−US;290497.

### 2.6.6. Virus Scanners

A number of different server−side virus scanners are available. To name a few:

- Sophie
- KAVDaemon
- ClamAV
- DrWeb

In situations where you are not willing to block all potentially dangerous files based on their file names alone (consider ".zip" files), such scanners are helpful. Also, they will be able to catch virii that are not transmitted as file attachments, such as the "Bagle.R" virus that arrived in March, 2004.

In most cases, the machine performing the virus scan does not need to be your mail exchanger. Most of these anti−virus scanners can be invoked on a different host over a network connection.

Anti−virus software mainly detect virii based on a set of signatures for known virii, or *virus definitions*. These need to be updated regularly, as new virii are developed. Also, the software itself should at any time be up to date for maximum accuracy.

### 2.6.7. Spam Scanners

Similarly, anti−spam software can be used to classify messages based on a large set of heuristics, including their content, standards compliance, and various network checks such as DNS Blacklists and Junk Mail Signature Repository. In the end, such software typically assigns a composite "score" to each message, indicating the likelihood that the message is spam, and if the score is above a certain threshold, would classify it as such.

Two of the most popular server−side heuristic anti−spam filters are:

- SpamAssassin
- BrightMail

These tools undergo a constant evolution as spammers find ways to circumvent their various checks. For instance, consider "creative" spelling, such as "GR0W lO 1NCH35". So, just like anti−virus software, if you use anti−spam software, you should update it frequently for the highest level of accuracy.

I use SpamAssassin, although to minimize impact on machine resources, it is no longer my first line of defense. Out of approximately 500 junk mail delivery attempts to my personal address per day, about 50 reach the point where they are being checked by SpamAssassin (mainly because they are forwarded from one of my other accounts, so the checks described above are not effective). Out of these 50 messages, one message ends up in my inbox approximately every 2 or 3 days.

## 2.7. Blocking Collateral Spam

*Collateral Spam* is more difficult to block with the techniques described so far, because it normally arrives from legitimate sites using standard mail transport software (such as Sendmail, Postfix, or Exim). The challenge is to distinguish these messages from valid *Delivery Status Notification*s returned in response to mail sent from your own users. Here are some ways that people do this:

### 2.7.1. Bogus Virus Warning Filter

Most of the time, collateral spam is virus warnings generated by anti−virus scanners[13]. In turn, the wording in the `Subject:` line of these virus warnings, and/or other characteristics, is usually provided by the anti−virus software itself. As such, you could create a list of the more common characteristics, and filter out such bogus virus warnings.

Well, aren't you in luck – someone already did this for you. :–)

Tim Jackson `<tim (at) timj.co.uk>` maintains a list of bogus virus warnings for use with SpamAssassin. This list is available at: http://www.timj.co.uk/linux/bogus–virus–warnings.cf.

## 2.7.2. Publish SPF info for your domain

The purpose of the Sender Policy Framework is precisely to protect against *Joe Job*s; i.e. to prevent forgeries of valid e–mail addresses.

If you publish SPF records in the DNS zone for your domain, then recipient hosts that incorporate SPF checks would not have accepted the forged message in the first place. As such, they would not be sending a *Delivery Status Notification* to your site.

## 2.7.3. Enveloper Sender Signature

A different approach that I am currently experimenting with myself is to add a signature in the local part of the *Envelope Sender* address in outgoing mail, then check for this signature in the *Envelope Recipient* address before accepting incoming *Delivery Status Notification*s. For instance, the generated sender address might be of the following format:

```
localpart=signature@domain
```

Normal message replies are unaffected. These replies go to the address in the `From:` or `Reply-To:` field of the message, which are left intact.

Sounds easy, doesn't it? Unfortunately, generating a signature that is suitable for this purpose is a bit more complex than it sounds. There are a couple of conflicting considerations to take into account:

- To gain any benefit from this method, the signed envelope sender address that you generate should be useless in the hands of spammers. Typically, this would imply that the signature incorporates a time stamp that would eventually expire:
  ```
  sender=timestamp=hash@domain
  ```
- If you send mail to a site that incorporates Greylisting, your envelope sender address should remain constant for that particular recipient. Otherwise, your mail will continuously be greylisted.

  With this in mind, you could generate a *Envelope Sender* based on the *Envelope Recipient* address:

  ```
  sender=recipient=recipient.domain=hash@domain
  ```

  Although this address does not expire, if you start seeing junk mail to it, you will at least know the source of the leak – it is incorported in the recipient address. Moreover, you can easily block specific recipient address signatures, without affecting normal mail delivery to that same recipient.
- Two more issues occur with mailing list servers. Usually, replies to request mails (such as "subscribe"/"unsubscribe") are sent with no envelope sender.

  - The first issue pertains to servers that send responses back to the *Envelope Sender* address of the request mail (as in the case of `<discuss@en.tldp.org>`). The problem is that commands for the mailing list server (such as **subscribe** or **unsubscribe**) are typically sent to one or more different addresses (e.g. `<discuss–subscribe@en.tldp.org>` and

`<discuss-unsubscribe@en.tldp.org>`, respectively) than the address used for list mail. Hence, the subscriber address will be different from the sender address in messages sent to the list itself –– and in this example, also different from the address that will be generated for unsubscription requests. As a result, you may not be able to post to the list, or unsubscribe.

The compromise would be to incorporate only the recipient *domain* in the sender signature. The sender address might then look like:

```
subscribername=en.tldp.org=hash@subscriber.domain
```

♦ The second issue pertains to those that send responses back to the reply address in the message header of the request mail (such as `<spam-l-request@peach.ease.lsoft.com>`). Since this address is not signed, the response from the list server would be blocked by your server.

  There is not much you can do about this, other than to "whitelist" these particular servers in such a way that they are allowed to return mail to unsigned recipient addresses.

At this point, this approach starts losing some of its edge. Moreover, even legitimate DSNs are rejected unless the original mail has been sent via your server. Thus, you should only consider doing this if for those of your users that do not roam, or otherwise send their outgoing mail via servers outside your control.

That said, in situations where none of the above concerns apply to you, this method gives you a good way to not only eliminate collateral spam, but also a way to educate the owners of the sites that (presumably unwittingly) generate it. Moreover, as a side benefit, sites that perform Sender Callout Verification will only get a positive response from you if the original mail was, indeed, sent from your site. In essence, you are reducing your exposure to sender address forgeries by spammers.

You could perhaps allow your users to specify whether to sign outgoing mails, and if so, specify which hosts should be allowed to return mails to the unsigned version of their address. For instance, if they have system accounts on your mail server, you could check for the existence and content, respectively, of a given file in their home directory.

## 2.7.4. Accept Bounces Only for Real Users

Even if you check for envelope sender signatures, there may be a loophole that allows bogus bounces to be accepted. Specifically, if your users have to opt in to the scheme, you are probably not checking for this signature in mails sent to system aliases, such as `postmaster` or `mailer-daemon`. Moreover, since these users do not generate outgoing mail, they should not receive any bounces.

You can reject mail if it is sent to such system aliases, or alternatively, if there is no mailbox for the provided recipient address.

# Chapter 3. Considerations

Some specific considerations come into play as a result of system−wide SMTP time filtering. Here we cover some of those.

## 3.1. Multiple Incoming Mail Exchangers

Most domains list more than one incoming *Mail Exchanger*s (a.k.a. "MX hosts"). If you do so, then bear in mind that in order to have any effect, any SMTP time filtering you incorporate on the primary MX has to be incorporated on all the others as well. Otherwise, the sending host would simply sidestep filtering by retrying the mail delivery through your backup server(s).

If the backup server(s) are not under your control, ask yourself whether you need multiple MXs in the first place. In this situation, chances are that they serve only as *redundant* mail servers, and that they in turn forward the mail to your primary MX. If so, you probably don't need them. If your host happens to be down for a little while, that's OK −− well−behaved sender hosts will retry deliveries for several days before giving up [9].

A situation where you *may* need multiple MXs is to perform load balancing between several servers – i.e. if you receive so much mail that one machine alone could not handle it. In this case, see if you could offload some tasks (such as virus and spam scanners) to other machines, in order to reduce or eliminate this need.

Again, if you do decide to keep using several MXs, your backup servers need to be (at least) as restrictive as the primary server, lest filtering in the primary MX is useless.

See also the section on Greylisting for additional concerns related to multiple MX hosts.

## 3.2. Blocking Access to Other SMTP Servers

Any SMTP server that is not listed as a public *Mail Exchanger* in the DNS zone of your domain(s) should not accept incoming connections from the internet. All incoming mail traffic should go through your incoming mail exchanger(s).

This consideration is not unique to SMTP servers. If you have machines that only serve an internal purpose within your site, use a firewall to restrict access to these.

This is a rule, so therefore there must be exceptions. However, if you don't know what they are, then the above applies to you.

## 3.3. Forwarded Mail

You should take care not to reject mail as a result of spam filtering if it is forwarded from "friendly" sources, such as:

- Your backup MX hosts, if any. Supposedly, these have already filtered out most of the junk (see Multiple Incoming Mail Exchangers).

* Mailing lists, to which you or your users subscribe. You may still filter such mail (it may not be as criticial if it ends up in a black hole). However, if you reject the mail, you may end up causing the list server to automatically unsubscribe the recipient.
* Other accounts belonging to the recipient. Again, rejections will generate collateral spam, and/or create problems for the host that forwards the mail.

You may see a logistical issue with the last two of these sources: They are specific to each recipient. How to you allow each user to specify which hosts they want to whitelist, and then use such individual whitelists in a system−wide SMTP−time filtering setup? If the message is forwarded to several recipients at your site (as may often be true in the case of a mailing list), how do you decide whose whitelist to use?

There is no magic bullet here. This is one of those situations where we just have to do a bit of work. You can decide to accept all mails, regardless of spam classification, so long as it is sent from a host in the whitelist of any one of the recipients. For instance, in response to each **RCPT TO:** command, we can match the sending host against the corresponding user's whitelist. If found, set a flag that will prevent a subsequent rejection. Effectively, you are using an *aggregate* of each recipient's whitelist.

The implementation appendices cover this in more detail.

## 3.4. User Settings and Data

There are other situations where you may want to support settings and data for each user at site. For instance, if you scan incoming mail with SpamAssassin (see Spam Scanners), you may want to allow for individual spam thresholds, acceptable languages and character sets, and Bayesian training/data.

A sticking point is that SMTP−time filtering of incoming mail is done at the system level, before mail is being delivered to a particular user, and as such, does not lend itself too well to individual preferences. A single message may have several recipients; and unlike the case with Forwarded Mail, using an aggregate of each recipient's preferences is not a good option. Consider a scenario where you have users from different linguistic backgrounds.

As it turns out, though, there is a modification to this truth. The trick is to limit the number of recipients in incoming messages to one, so that the message can be analyzed in accordance with the settings and data that belongs to the corresponding user.

To do this, you would accept the first **RCPT TO:**, then issue a SMTP **451** (defer) response to subsequent commands. If the caller is a well−behaved MTA, it will know how to interpret this response, and try later. (If it is confused, then, well, it is probably a sender from which you don't want to receive mail in the first place).

Obviously, this is a hack. Every mail sent to several users at your site will be slowed down by 30 minutes or more per recipient. Especially in corporate environments, where it is common to see e−mail discussions involving several people on the inside and several others on the outside, and where timelines of mail deliveries are essential, this is probably not a good solution at all.

Another issue that mainly pertains to corporate enterprises and other large sites is that incoming mail is often forwarded to internal machines for delivery, and that recipients don't normally have accounts on the mail exchanger. It may still be possible to support user−specific settings and data in these situations (e.g. via database lookups or LDAP queries), but you may also want to consider whether it's worth the effort.

That said, if you are on a small site, and where you are not afraid of delayed deliveries, this may be an acceptable way to allow each user to fine tune their filtering criteria.

---

# Chapter 4. Questions & Answers

In this section I try to anticipate some of the questions that may come up, and to answer them. If you have questions that are not listed, and/or would like to provide extra input in this section, please provide feedback.

## When Spammers Adapt

*Q: What happens when spammers adapt and try to get around the techniques described in this document?*

**Q:** What happens when spammers adapt and try to get around the techniques described in this document?

**A:** Well, that depends. :−)

Some of the checks described (such as SMTP checks and Greylisting) specifically target *ratware* behavior. It is certainly possible to imagine that this behavior will change if enough sites incorporate these checks. Hatmut Danisch notes: *Ratware contains buggy SMTP protocols because they didn't need to do any better. It worked this way, so why should they have spent more time? Meanwhile "ratware" has a higher quality, and even the quality of spam messages has significantly improved. Once enough people reject spam by detecting bad SMTP protocols, spam software authors will simply improve their software.*

That said, there are challenges remaining for such ratware:

- To get around SMTP transaction delays, they need to wait for each response from the receiving SMTP server. At that point, we have collectively accomplished a significant reduction in the rate of mail that a given spamming host is able to deliver per unit of time. Since spammers are racing against time to deliver as many mails as possible before DNS blocklists and collaborative content filters catch up, we are improving the effectiveness of these tools.

  The effect is similar to the goal of *Micropayment Schemes*, wherein the sender spends a few seconds working on a computational challenge for each recipient of the mail, and adds a resulting signature to the e−mail header for the recipient to validate. The main difference, aside from the complexity of these schemes, is that they require the participation of virtually everyone in the world before they can effectively be used to weed out spam, whereas SMTP transaction delays start being effective with the first recipient machine that implements it.
- To get around a HELO/EHLO check, they need to provide a proper greeting, i.e. identify themselves with a valid *Fully Qualified Domain Name*. This provides for increased traceability, especially with receiving *Mail Transport Agent*s that do not automatically insert the results of a rDNS lookup into the Received: header of the message.
- To get all of the Sender Address Checks, they need to provide their own valid sender address (or, at least, *a* valid sender address within their own domain). Nuff said.
- To get around Greylisting, they need to retry deliveries to temporarily failed recipients addresses after one hour (but before four hours). (As far as implementation goes, in order to minimize machine resources, rather than keeping a copy of each temporarily failed mail, ratware may keep only a list of temporarily failed recipients, and perform a second sweep through those addresses after an hour or two).

  Even so, *greylisting* will remain fairly effective in conjunction with DNS Blacklists that are fed from *Spam Trap*s. That is because the mandatory one−hour retry delay will give these lists a chance to list the sending host.

Software tools, such as Spam Scanners and Virus Scanners, are in constant evolution. As spammers evolve, so do these (and vice versa). As long as you use recent versions of these tools, they will remain quite effective.

Finally, this document is itself subject to change. As the nature of junk mail changes, people will come up with new, creative ways to block it.

# Appendix A. Exim Implementation

Here we cover the integration of techniques and tools described in this document into the Exim *Mail Transport Agent*.

## A.1. Prerequisites

For these examples, you need the `Exim` *Mail Transport Agent*, preferrably with Tom Kistner's `Exiscan−ACL` patch applied. Prebuilt `Exim+Exiscan−ACL` packages exist for the most popular Linux distributions as well as FreeBSD; see the Exiscan−ACL home page for details[14].

The final implementation example at the end incorporates these additional tools:

- SpamAssassin − a popular spam filtering tool that analyzes mail content against a large and highly sophisticated set of heuristics.
- greylistd − a simple greylisting solution written by yours truly, specifically with Exim in mind.

Other optional software is used in examples throughout.

## A.2. The Exim Configuration File

The Exim configuration file contains global definitions at the top (we will call this the *main section*), followed by several other sections[15]. Each of these other sections starts with:

```
begin section
```

We will spend most of our time in the `acl` section (i.e. after `begin acl`); but we will also add and/or modify a few items in the `transports` and `routers` sections, as well as in the main section at the top of the file.

### A.2.1. Access Control Lists

As of version 4.xx, Exim incorporates perhaps the most sophisticated and flexible mechanism for SMTP−time filtering available anywhere, by way of so−called *Access Control Lists* (ACLs).

An ACL can be used to evaluate whether to accept or reject an aspect of an incoming message transaction, such as the initial connection from a remote host, or the **HELO/EHLO**, **MAIL FROM:**, or **RCPT TO:** SMTP commands. So, for instance, you may have an ACL named `acl_rcpt_to` to validate each **RCPT TO:** command received from the peer.

An ACL consists of a series of *statements* (or *rules*). Each statement starts with an action verb, such as `accept`, `warn`, `require`, `defer`, or `deny`, followed by a list of conditions, options, and other settings pertaining to that statement. Every *statement* is evaluated in order, until a definitive action (besides `warn`) is taken. There is an implicit `deny` at the end of the ACL.

A sample statement in the `acl_rcpt_to` ACL above may look like this:

```
  deny
    message  = relay not permitted
    !hosts   = +relay_from_hosts
    !domains = +local_domains : +relay_to_domains
    delay    = 1m
```

This statement will reject the **RCPT TO:** command if it was not delivered by a host in the "+relay_from_hosts" host list, and the recipient domain is not in the "+local_domains" or "+relay_to_domains" domain lists. However, before issuing the "550" SMTP response to this command, the server will wait for one minute.

To evaluate a particular ACL at a given stage of the message transaction, you need to point one of Exim's *policy controls* to that ACL. For instance, to use the `acl_rcpt_to` ACL mentioned above to evaluate the **RCPT TO:**, the main section of your Exim configuration file (before any `begin` keywords) should include:

```
acl_smtp_rcpt = acl_rcpt_to
```

For a full list of such *policy controls*, refer to section 14.11 in the Exim specifications.

### A.2.2. Expansions

A large number of *expansion items* are available, including run−time variables, lookup functions, string/regex manipulations, host/domain lists, etc. etc. An exhaustive reference for the last x.x0 release (i.e. 4.20, 4.30..) can be found in the file "spec.txt"; ACLs are described in section 38.

In particular, Exim provides twenty general purpose expansion variables to which we can assign values in an ACL statement:

- `$acl_c0` – `$acl_c9` can hold values that will persist through the lifetime of an SMTP connection.
- `$acl_m0` – `$acl_m9` can hold values while a message is being received, but are then reset. They are also reset by the **HELO**, **EHLO**, **MAIL**, and **RSET** commands.

## A.3. Options and Settings

The main section of the Exim configuration file (before the first `begin` keyword) contains various macros, policy controls, and other general settings. Let us start by defining a couple of macros we will use later:

```
# Define the message size limit; we will use this in the DATA ACL.
MESSAGE_SIZE_LIMIT = 10M

# Maximum message size for which we will run Spam or Virus scanning.
# This is to reduce the load imposed on the server by very large messages.
MESSAGE_SIZE_SPAM_MAX = 1M

# Macro defining a secret that we will use to generate various hashes.
# PLEASE CHANGE THIS!.
SECRET = some-secret
```

Let us tweak some general Exim settings:

```
# Treat DNS failures (SERVFAIL) as lookup failures.
# This is so that we can later reject sender addresses
```

```
# within non-existing domains, or domains for which no
# nameserver exists.
dns_again_means_nonexist = !+local_domains : !+relay_to_domains

# Enable HELO verification in ACLs for all hosts
helo_try_verify_hosts = *

# Remove any limitation on the maximum number of incoming
# connections we can serve at one time.  This is so that while
# we later impose SMTP transaction delays for spammers, we
# will not refuse to serve new connections.
smtp_accept_max = 0

# ..unless the system load is above 10
smtp_load_reserve = 10

# Do not advertise ESMTP "PIPELINING" to any hosts.
# This is to trip up ratware, which often tries to pipeline
# commands anyway.
pipelining_advertise_hosts = :
```

Finally, we will point some Exim policy controls to five ACLs that we will create to evaluate the various stages of an incoming SMTP transaction:

```
acl_smtp_connect = acl_connect
acl_smtp_helo    = acl_helo
acl_smtp_mail    = acl_mail_from
acl_smtp_rcpt    = acl_rcpt_to
acl_smtp_data    = acl_data
```

## A.4. Building the ACLs – First Pass

In the acl section (following `begin acl`), we need to define these ACLs. In doing so, we will incorporate some of the basic *Techniques* described earlier in this document, namely *DNS checks* and *SMTP checks*.

In this pass, we will do most of the checks in acl_rcpt_to, and leave the other ACLs largely empty. That is because most of the commonly used ratware does not understand rejections early in the SMTP transaction – it keeps trying. On the other hand, most ratware clients give up if the **RCPT TO:** fails.

We create all these ACLs, however, because we will use them later.

### A.4.1. acl_connect

```
# This access control list is used at the start of an incoming
# connection.  The tests are run in order until the connection
# is either accepted or denied.

acl_connect:

  # In this pass, we do not perform any checks here.
  accept
```

## A.4.2. acl_helo

```
# This access control list is used for the HELO or EHLO command in
# an incoming SMTP transaction.  The tests are run in order until the
# greeting is either accepted or denied.

acl_helo:

  # In this pass, we do not perform any checks here.
  accept
```

## A.4.3. acl_mail_from

```
# This access control list is used for the MAIL FROM: command in an
# incoming SMTP transaction.  The tests are run in order until the
# sender address is either accepted or denied.
#

acl_mail_from:

  # Accept the command.
  accept
```

## A.4.4. acl_rcpt_to

```
# This access control list is used for every RCPT command in an
# incoming SMTP message.  The tests are run in order until the
# recipient address is either accepted or denied.

acl_rcpt_to:

  # Accept mail received over local SMTP (i.e. not over TCP/IP).
  # We do this by testing for an empty sending host field.
  # Also accept mails received from hosts for which we relay mail.
  #
  # Recipient verification is omitted here, because in many
  # cases the clients are dumb MUAs that don't cope well with
  # SMTP error responses.
  #
  accept
    hosts       = : +relay_from_hosts


  # Accept if the message arrived over an authenticated connection,
  # from any host. Again, these messages are usually from MUAs, so
  # recipient verification is omitted.
  #
  accept
    authenticated = *


  ######################################################################
  # DNS checks
  ######################################################################
  #
  # The results of these checks are cached, so multiple recipients
  # does not translate into multiple DNS lookups.
  #
```

```
# If the connecting host is in one of a select few DNSbls, then
# reject the message.  Be careful when selecting these lists; many
# would cause a large number of false postives, and/or have no
# clear removal policy.
#
deny
  dnslists    = dnsbl.sorbs.net : \
                dnsbl.njabl.org : \
                cbl.abuseat.org : \
                bl.spamcop.net
  message     = $sender_host_address is listed in $dnslist_domain\
                ${if def:dnslist_text { ($dnslist_text)}}


# If reverse DNS lookup of the sender's host fails (i.e. there is
# no rDNS entry, or a forward lookup of the resulting name does not
# match the original IP address), then reject the message.
#
deny
  message     = Reverse DNS lookup failed for host $sender_host_address.
  !verify     = reverse_host_lookup



######################################################################
# Hello checks
######################################################################

# If the remote host greets with an IP address, then reject the mail.
#
deny
  message     = Message was delivered by ratware
  log_message = remote host used IP address in HELO/EHLO greeting
  condition   = ${if isip {$sender_helo_name}{true}{false}}


# Likewise if the peer greets with one of our own names
#
deny
  message     = Message was delivered by ratware
  log_message = remote host used our name in HELO/EHLO greeting.
  condition   = ${if match_domain{$sender_helo_name}\
                     {$primary_hostname:+local_domains:+relay_to_domains}\
                     {true}{false}}


deny
  message     = Message was delivered by ratware
  log_message = remote host did not present HELO/EHLO greeting.
  condition   = ${if def:sender_helo_name {false}{true}}


# If HELO verification fails, we add a X-HELO-Warning: header in
# the message.
#
warn
  message     = X-HELO-Warning: Remote host $sender_host_address \
                ${if def:sender_host_name {($sender_host_name) }}\
                incorrectly presented itself as $sender_helo_name
  log_message = remote host presented unverifiable HELO/EHLO greeting.
  !verify     = helo
```

```
######################################################################
# Sender Address Checks
######################################################################

# If we cannot verify the sender address, deny the message.
#
# You may choose to remove the "callout" option.  In particular,
# if you are sending outgoing mail through a smarthost, it will not
# give any useful information.
#
# Details regarding the failed callout verification attempt are
# included in the 550 response; to omit these, change
# "sender/callout" to "sender/callout,no_details".
#
deny
  message     = <$sender_address> does not appear to be a \
                valid sender address.
  !verify     = sender/callout



######################################################################
# Recipient Address Checks
######################################################################

# Deny if the local part contains @ or % or / or | or !. These are
# rarely found in genuine local parts, but are often tried by people
# looking to circumvent relaying restrictions.
#
# Also deny if the local part starts with a dot. Empty components
# aren't strictly legal in RFC 2822, but Exim allows them because
# this is common.  However, actually starting with a dot may cause
# trouble if the local part is used as a file name (e.g. for a
# mailing list).
#
deny
  local_parts = ^.*[@%!/|] : ^\\.


# Drop the connection if the envelope sender is empty, but there is
# more than one recipient address.  Legitimate DSNs are never sent
# to more than one address.
#
drop
  message      = Legitimate bounces are never sent to more than one \
                 recipient.
  senders      = : postmaster@*
  condition    = $recipients_count


# Reject the recipient address if it is not in a domain for
# which we are handling mail.
#
deny
  message     = relay not permitted
  !domains    = +local_domains : +relay_to_domains


# Reject the recipient if it is not a valid mailbox.
```

```
  # If the mailbox is not on our system (e.g. if we are a
  # backup MX for the recipient domain), then perform a
  # callout verification; but if the destination server is
  # not responding, accept the recipient anyway.
  #
  deny
    message     = unknown user
    !verify     = recipient/callout=20s,defer_ok


  # Otherwise, the recipient address is OK.
  #
  accept
```

## A.4.5. acl_data

```
# This access control list is used for message data received via
# SMTP.  The tests are run in order until the recipient address
# is either accepted or denied.

acl_data:

  # Add Message-ID if missing in messages received from our own hosts.
  warn
    condition   = ${if !def:h_Message-ID: {1}}
    hosts       = : +relay_from_hosts
    message     = Message-ID: <E$message_id@$primary_hostname>


  # Accept mail received over local SMTP (i.e. not over TCP/IP).
  # We do this by testing for an empty sending host field.
  # Also accept mails received from hosts for which we relay mail.
  #
  accept
    hosts       = : +relay_from_hosts

  # Accept if the message arrived over an authenticated connection, from
  # any host.
  #
  accept
    authenticated = *


  # Enforce a message-size limit
  #
  deny
    message     = Message size $message_size is larger than limit of \
                  MESSAGE_SIZE_LIMIT
    condition   = ${if >{$message_size}{MESSAGE_SIZE_LIMIT}{true}{false}}


  # Deny unless the address list header is syntactically correct.
  #
  deny
    message     = Your message does not conform to RFC2822 standard
    log_message = message header fail syntax check
    !verify     = header_syntax


  # Deny non-local messages with no Message-ID, or no Date
```

```
#
# Note that some specialized MTAs, such as certain mailing list
# servers, do not automatically generate a Message-ID for bounces.
# Thus, we add the check for a non-empty sender.
#
deny
  message     = Your message does not conform to RFC2822 standard
  log_message = missing header lines
  !hosts      = +relay_from_hosts
  !senders    = : postmaster@*
  condition   = ${if or {{!def:h_Message-ID:}\
                         {!def:h_Date:}\
                         {!def:h_Subject:}} {true}{false}}


# Warn unless there is a verifiable sender address in at least
# one of the "Sender:", "Reply-To:", or "From:" header lines.
#
warn
  message     = X-Sender-Verify-Failed: No valid sender in message header
  log_message = No valid sender in message header
  !verify     = header_sender


# Accept the message.
#
accept
```

## A.5. Adding SMTP transaction delays

### A.5.1. The simple way

The simplest way to add SMTP transaction delays is to append a `delay` control to the final `accept` statement in each of the ACLs we have declared, as follows:

```
accept
  delay = 20s
```

In addition, you may want to add progressive delays in the `deny` statement pertaining to invalid recipients ("unknown user") within acl_rcpt_to. This is to slow down dictionary attacks. For instance:

```
deny
  message     = unknown user
  !verify     = recipient/callout=20s,defer_ok,use_sender
  delay       = ${eval:$rcpt_fail_count*10 + 20}s
```

It should be noted that there is no point in imposing a delay in acl_data, after the message data has been received. Ratware commonly disconnect at this point, before even receiving a response from your server. In any case, whether or not the client disconnects at this point has no bearing on whether Exim will proceed with the delivery of the message.

## A.5.2. Selective Delays

If you are like me, you want to be a little bit more selective about which hosts you subject to SMTP transaction delays. For instance, as described earlier in this document, you may decide that a match from a DNS blacklist or a non−verifiable EHLO/HELO greeting are not conditions that by themselves warrant a rejection − but they may well be sufficient triggers for transaction delays.

In order perform selective delays, we want move some of the checks that we previously did in acl_rcpt_to to earlier points in the SMTP transaction. This is so that we can start imposing the delays as soon as we see any sign of trouble, and thereby increase the chance of causing synchronization errors and other trouble for ratware.

Specifically, we want to:

- Move the DNS checks to acl_connect.
- Move the Hello checks to acl_helo. One exception: We cannot yet check for a missing Hello greeting at this point, because this ACL is processed *in response* to an EHLO or HELO command. We will do this check in the acl_mail_from ACL.
- Move the Sender Address Checks checks to acl_mail_from.

However, for reasons described above, we do not want to actually reject the mail until after the **RCPT TO:** command. Instead, in the earlier ACLs, we will convert the various `deny` statements into `warn` statements, and use Exim's general purpose ACL variables to store any error messages or warnings until after the **RCPT TO:** command. We do that as follows:

- If we decide to reject the delivery, we store an error message to be used in the forthcoming **550** response in `$acl_c0` or `$acl_m0`:
    - ♦ If we identify the condition before a mail delivery has started (i.e. in acl_connect or acl_helo), we use the connection−persistent variable `$acl_c0`
    - ♦ Once a mail transaction has started (i.e. after the **MAIL FROM:** command), we copy any contents from `$acl_c0` into the message−specific variable `$acl_m0`, and use the latter from this point forward. This way, any conditions identified in this particular message will not affect any subsequent messages received in the same connection.

  Also, we store a corresponding *log message* in `$acl_c1` or `$acl_m1`, in a similar manner.
- If we come across a condition that does not warrant an outright rejection, we only store a warning message in `$acl_c1` or `$acl_m1`. Once a mail transaction has started (i.e. in acl_mail_from), we add any content in this variable to the message header as well.
- If we decide to *accept* a message without regard to the results of any subsequent checks (such as a SpamAssassin scan), we set a flag in `$acl_c0` or `$acl_m0`, but `$acl_c1` and `$acl_m1` empty.
- At the beginning of every ACL to and including acl_mail_from, we record the current timestamp in `$acl_m2`. At the end of the ACL, we use the presence of `$acl_c1` or `$acl_m1` to trigger a SMTP transaction delay until a total of 20 seconds has elapsed.

The following table summarizes our use of these variables:

**Table A−1. Use of ACL connection/message variables**

| Variables: | $acl_[cm]0 unset | $acl_[cm]0 set |
|---|---|---|

| | | |
|---|---|---|
| $acl_[cm]1 unset | (No decision yet) | Accept the mail |
| $acl_[cm]1 set | Add warning in header | Reject the mail |

As an example of this approach, let us consider two checks that we do in response to the Hello greeting; one that will reject mails if the peer greets with an IP address, and one that will warn about an unverifiable name in the greeting. Previously, we did both of these checks in acl_rcpt_to – now we move them to the acl_helo ACL.

```
acl_helo:
  # Record the current timestamp, in order to calculate elapsed time
  # for subsequent delays
  warn
    set acl_m2  = $tod_epoch


  # Accept mail received over local SMTP (i.e. not over TCP/IP).
  # We do this by testing for an empty sending host field.
  # Also accept mails received from hosts for which we relay mail.
  #
  accept
    hosts       = : +relay_from_hosts


  # If the remote host greets with an IP address, then prepare a reject
  # message in $acl_c0, and a log message in $acl_c1.  We will later use
  # these in a "deny" statement.  In the mean time, their presence indicate
  # that we should keep stalling the sender.
  #
  warn
    condition   = ${if isip {$sender_helo_name}{true}{false}}
    set acl_c0  = Message was delivered by ratware
    set acl_c1  = remote host used IP address in HELO/EHLO greeting


  # If HELO verification fails, we prepare a warning message in acl_c1.
  # We will later add this message to the mail header.  In the mean time,
  # its presence indicates that we should keep stalling the sender.
  #
  warn
    condition   = ${if !def:acl_c1 {true}{false}}
    !verify     = helo
    set acl_c1  = X-HELO-Warning: Remote host $sender_host_address \
                  ${if def:sender_host_name {($sender_host_name) }}\
                  incorrectly presented itself as $sender_helo_name
    log_message = remote host presented unverifiable HELO/EHLO greeting.


  #
  # ... additional checks omitted for this example ...
  #


  # Accept the connection, but if we previously generated a message in
  # $acl_c1, stall the sender until 20 seconds has elapsed.
  accept
    set acl_m2  = ${if def:acl_c1 {${eval:20 + $acl_m2 - $tod_epoch}}{0}}
    delay       = ${if >{$acl_m2}{0}{$acl_m2}{0}}s
```

Then, in acl_mail_from we transfer the messages from `$acl_c{0,1}` to `$acl_m{0,1}`. We also add the contents of `$acl_c1` to the message header.

```
acl_mail_from:
  # Record the current timestamp, in order to calculate elapsed time
  # for subsequent delays
  warn
    set acl_m2  = $tod_epoch


  # Accept mail received over local SMTP (i.e. not over TCP/IP).
  # We do this by testing for an empty sending host field.
  # Also accept mails received from hosts for which we relay mail.
  #
  accept
    hosts     = : +relay_from_hosts


  # If present, the ACL variables $acl_c0 and $acl_c1 contain rejection
  # and/or warning messages to be applied to every delivery attempt in
  # in this SMTP transaction.  Assign these to the corresponding
  # $acl_m{0,1} message-specific variables, and add any warning message
  # from $acl_m1 to the message header.  (In the case of a rejection,
  # $acl_m1 actually contains a log message instead, but this does not
  # matter, as we will discard the header along with the message).
  #
  warn
    set acl_m0  = $acl_c0
    set acl_m1  = $acl_c1
    message     = $acl_c1


  #
  # ... additional checks omitted for this example ...
  #

  # Accept the sender, but if we previously generated a message in
  # $acl_c1, stall the sender until 20 seconds has elapsed.
  accept
    set acl_m2  = ${if def:acl_c1 {${eval:20 + $acl_m2 - $tod_epoch}}{0}}
    delay       = ${if >{$acl_m2}{0}{$acl_m2}{0}}s
```

All the pertinent changes are incorporated in the Final ACLs, to follow.

## A.6. Adding Greylisting Support

There are several alternate greylisting implementations available for Exim. Here we will cover a couple of these.

### A.6.1. greylistd

This is a Python implementation developed by *yours truly*. (So naturally, this is the implementation I will include in the Final ACLs to follow). It operates as a stand–alone daemon, and thus does not depend on any external database. Greylist data is stored as simple 32–bit hashes for efficiency.

You can find it at http://packages.debian.org/unstable/mail/greylistd. Debian users can get it via APT:

```
# apt-get install greylistd
```

To consult `greylistd`, we insert two statements in acl_rcpt_to ACL that we previously declared, right before the final `accept` statement:

```
  # Consult "greylistd" to obtain greylisting status for this particular
  # peer/sender/recipient triplet.
  #
  # We do not greylist messages with a NULL sender, because sender
  # callout verification would break (and we might not be able to
  # send mail to a host that performs callouts).
  #
  defer
    message     = $sender_host_address is not yet authorized to deliver mail \
                  from <$sender_address> to <$local_part@$domain>. \
                  Please try later.
    log_message = greylisted.
    domains     = +local_domains : +relay_to_domains
    !senders    = : postmaster@*
    set acl_m9  = $sender_host_address $sender_address $local_part@$domain
    set acl_m9  = ${readsocket{/var/run/greylistd/socket}{$acl_m9}{5s}{}{}}
    condition   = ${if eq {$acl_m9}{grey}{true}{false}}
```

Unless you incorporate envelope sender signatures to block bogus *Delivery Status Notification*s, you may want to add a similar statement in your acl_data to also greylist messages with a NULL sender.

The data we use for greylisting purposes here will be a little different than above. In addition to `$sender_address` being emtpy, neither `$local_part` nor `$domain` is defined at this point. Instead, the variable `$recipients` contains a comma–separated list of all recipient addresses. For a legitimate DSN, there should be only one address.

```
  # Perform greylisting on messages with no envelope sender here.
  # We did not subject these to greylisting after RCPT TO: because
  # that would interfere with remote hosts doing sender callouts.
  #
  defer
    message     = $sender_host_address is not yet authorized to send \
                  delivery status reports to <$recipients>. \
                  Please try later.
    log_message = greylisted.
    senders     = : postmaster@*
    set acl_m9  = $sender_host_address $recipients
    set acl_m9  = ${readsocket{/var/run/greylistd/socket}{$acl_m9}{5s}{}{}}
    condition   = ${if eq {$acl_m9}{grey}{true}{false}}
```

## A.6.2. MySQL implementation

The following inline implementation was contributed by Johannes Berg `<johannes (at) sipsolutions.net>`, based in part on:

- work by Rick Stewart `<rick.stewart (at) theinternetco.net>`, published at http://theinternetco.net/projects/exim/greylist, in turn based on
- a Postgres implementation created by Tollef Fog Heen `<tfheen (at) raw.no>`, available at

http://raw.no/personal/blog/tech/Debian/2004−03−14−15−55_greylisting

It requires no external programs − the entire implementation is based on these configuration snippets along with a MySQL database.

An archive containing up−to−date configuration snippets as well as a README file is available at: http://johannes.sipsolutions.net/wiki/Projects/exim−greylist.

MySQL needs to be installed on your system. At a MySQL prompt, create an exim4 database with two tables named exim_greylist and exim_greylist_log, as follows:

```
CREATE DATABASE exim4;
use exim4;

CREATE TABLE exim_greylist (
   id bigint(20) NOT NULL auto_increment,
   relay_ip varchar(80) default NULL,
   sender varchar(255) default NULL,
   recipient varchar(255) default NULL,
   block_expires datetime NOT NULL default '0000-00-00 00:00:00',
   record_expires datetime NOT NULL default '9999-12-31 23:59:59',
   create_time datetime NOT NULL default '0000-00-00 00:00:00',
   type enum('AUTO','MANUAL') NOT NULL default 'MANUAL',
   passcount bigint(20) NOT NULL default '0',
   blockcount bigint(20) NOT NULL default '0',
   PRIMARY KEY  (id)
);

CREATE TABLE exim_greylist_log (
   id bigint(20) NOT NULL auto_increment,
   listid bigint(20) NOT NULL,
   timestamp datetime NOT NULL default '0000-00-00 00:00:00',
   kind enum('deferred', 'accepted') NOT NULL,
   PRIMARY KEY (id)
);
```

In the *main* section of your Exim configuration file, declare the following macros:

```
# if you don't have another database defined, then define it here
hide mysql_servers = localhost/exim4/user/password

# options
# these need to be valid as xxx in mysql's DATE_ADD(..,INTERVAL xxx)
# not valid, for example, are plurals: "2 HOUR" instead of "2 HOURS"
GREYLIST_INITIAL_DELAY = 1 HOUR
GREYLIST_INITIAL_LIFETIME = 4 HOUR
GREYLIST_WHITE_LIFETIME = 36 DAY
GREYLIST_BOUNCE_LIFETIME = 0 HOUR

# you can change the table names
GREYLIST_TABLE=exim_greylist
GREYLIST_LOG_TABLE=exim_greylist_log

# comment out to the following line to disable greylisting (temporarily)
GREYLIST_ENABLED=

# uncomment the following to enable logging
#GREYLIST_LOG_ENABLED=
```

```
# below here, nothing should normally be edited

.ifdef GREYLIST_ENABLED
# database macros
GREYLIST_TEST = SELECT CASE \
   WHEN now() > block_expires THEN "accepted" \
   ELSE "deferred" \
 END AS result, id \
 FROM GREYLIST_TABLE \
 WHERE (now() < record_expires) \
   AND (sender      = '${quote_mysql:$sender_address}' \
        OR (type='MANUAL' \
            AND (    sender IS NULL \
                  OR sender = '${quote_mysql:@$sender_address_domain}' \
                ) \
           ) \
       ) \
   AND (recipient   = '${quote_mysql:$local_part@$domain}' \
        OR (type = 'MANUAL' \
            AND (    recipient IS NULL \
                  OR recipient = '${quote_mysql:$local_part@}' \
                  OR recipient = '${quote_mysql:@$domain}' \
                ) \
           ) \
       ) \
   AND (relay_ip    = '${quote_mysql:$sender_host_address}' \
        OR (type='MANUAL' \
            AND (    relay_ip IS NULL \
                  OR relay_ip = substring('${quote_mysql:$sender_host_address}',1,length(relay_ip))
                ) \
           ) \
       ) \
 ORDER BY result DESC LIMIT 1

GREYLIST_ADD = INSERT INTO GREYLIST_TABLE \
  (relay_ip, sender, recipient, block_expires, \
   record_expires, create_time, type) \
 VALUES ( '${quote_mysql:$sender_host_address}', \
  '${quote_mysql:$sender_address}', \
  '${quote_mysql:$local_part@$domain}', \
  DATE_ADD(now(), INTERVAL GREYLIST_INITIAL_DELAY), \
  DATE_ADD(now(), INTERVAL GREYLIST_INITIAL_LIFETIME), \
  now(), \
  'AUTO' \
)

GREYLIST_DEFER_HIT = UPDATE GREYLIST_TABLE \
                     SET blockcount=blockcount+1 \
                     WHERE id = $acl_m9

GREYLIST_OK_COUNT = UPDATE GREYLIST_TABLE \
                    SET passcount=passcount+1 \
                    WHERE id = $acl_m9

GREYLIST_OK_NEWTIME = UPDATE GREYLIST_TABLE \
                      SET record_expires = DATE_ADD(now(), INTERVAL GREYLIST_WHITE_LIFETIME) \
                      WHERE id = $acl_m9 AND type='AUTO'

GREYLIST_OK_BOUNCE = UPDATE GREYLIST_TABLE \
                     SET record_expires = DATE_ADD(now(), INTERVAL GREYLIST_BOUNCE_LIFETIME) \
                     WHERE id = $acl_m9 AND type='AUTO'
```

```
GREYLIST_LOG = INSERT INTO GREYLIST_LOG_TABLE \
               (listid, timestamp, kind) \
               VALUES ($acl_m9, now(), '$acl_m8')
.endif
```

Now, in the ACL section (after `begin acl`), declare a new ACL named "greylist_acl":

```
.ifdef GREYLIST_ENABLED
# this acl returns either deny or accept
# since we use it inside a defer with acl = greylist_acl,
# accepting here makes the condition TRUE thus deferring,
# denying here makes the condition FALSE thus not deferring
greylist_acl:
  # For regular deliveries, check greylist.

  # check greylist tuple, returning "accepted", "deferred" or "unknown"
  # in acl_m8, and the record id in acl_m9

  warn set acl_m8 = ${lookup mysql{GREYLIST_TEST}{$value}{result=unknown}}
       # here acl_m8 = "result=x id=y"

       set acl_m9 = ${extract{id}{$acl_m8}{$value}{-1}}
       # now acl_m9 contains the record id (or -1)

       set acl_m8 = ${extract{result}{$acl_m8}{$value}{unknown}}
       # now acl_m8 contains unknown/deferred/accepted

  # check if we know a certain triple, add and defer message if not
  accept
       # if above check returned unknown (no record yet)
       condition = ${if eq{$acl_m8}{unknown}{1}}
       # then also add a record
       condition = ${lookup mysql{GREYLIST_ADD}{yes}{no}}

  # now log, no matter what the result was
  # if the triple was unknown, we don't need a log entry
  # (and don't get one) because that is implicit through
  # the creation time above.
  .ifdef GREYLIST_LOG_ENABLED
  warn condition = ${lookup mysql{GREYLIST_LOG}}
  .endif

  # check if the triple is still blocked
  accept
       # if above check returned deferred then defer
       condition = ${if eq{$acl_m8}{deferred}{1}}
       # and note it down
       condition = ${lookup mysql{GREYLIST_DEFER_HIT}{yes}{yes}}

  # use a warn verb to count records that were hit
  warn condition = ${lookup mysql{GREYLIST_OK_COUNT}}

  # use a warn verb to set a new expire time on automatic records,
  # but only if the mail was not a bounce, otherwise set to now().
  warn !senders = : postmaster@*
       condition = ${lookup mysql{GREYLIST_OK_NEWTIME}}
  warn senders = : postmaster@*
       condition = ${lookup mysql{GREYLIST_OK_BOUNCE}}

  deny
.endif
```

Incorporate this ACL into your acl_rcpt_to to greylist triplets where the sender address is non−empty. This is to allow for sender callout verifications:

```
.ifdef GREYLIST_ENABLED
  defer !senders = : postmaster@*
        acl      = greylist_acl
        message  = greylisted - try again later
.endif
```

Also incorporate it into your acl_data block, but this time only if the sender address is empty. This is to prevent spammers from getting around greylisting by setting the sender address to NULL.

```
.ifdef GREYLIST_ENABLED
  defer senders  = : postmaster@*
        acl      = greylist_acl
        message  = greylisted - try again later
.endif
```

# A.7. Adding SPF Checks

Here we cover two different ways to check Sender Policy Framework records using Exim. In addition to these explicit mechanisms, the SpamAssassin suite will in the near future (around version 2.70) incorporate more sophisticated SPF checks, by assigning weighted scores to the various SPF results.

Although we *could* perform this check as early as in the acl_mail_from ACL, there is an issue that will affect this decision: SPF is incompatible with traditional e−mail forwarding. Unless the forwarding host implements SRS, you may end up rejecting forwarded mail because you receive it from a host that is not authorized to do so per the SPF policy of the domain in the *Envelope Sender* address.

To avoid doing this, we need to consult a user−specific list of hosts from which forwarded mails should be accepted (as described in Exempting Forwarded Mail, to follow). This is only possible after the **RCPT TO:**, when we know the username of the recipient.

As such, we will add this check prior to any greylisting checks and/or the final `accept` statement in acl_rcpt_to.

## A.7.1. SPF checks via Exiscan−ACL

Recent versions of Tom Kistner's `Exiscan-ACL` patch (see Prerequisites) have native support for SPF. [16] Usage is very simple. An `spf` ACL condition is added, and can be compared against any of the keywords `pass`, `fail`, `softfail`, `none`, `neutral`, `err_perm` or `err_temp`.

Prior to any greylisting checks and/or the final `accept` statement in acl_rcpt_to, insert the following snippet:

```
  # Query the SPF information for the sender address domain, if any,
  # to see if the sending host is authorized to deliver its mail.
  # If not, reject the mail.
  #
  deny
    message     = [SPF] $sender_host_address is not allowed to send mail \
                  from $sender_address_domain
    log_message = SPF check failed.
    spf         = fail
```

```
# Add a SPF-Received: header to the message
warn
  message     = $spf_received
```

This statement will reject the mail if the owner of the domain in the sender address has disallowed deliveries from the calling host. Some people find that this gives the domain owner a little bit too much control, even to the point of shooting themselves in the foot. A suggested alternative is to combine the SPF check with other checks, such as Sender Callout Verification (but note that as before, there is no point in doing this if you are sending your outgoing mail through a smarthost):

```
# Reject the mail if we cannot verify the sender address via callouts,
# and if SPF information for the sending domain does not grant explicit
# authority to the sending host.
#
deny
  message     = The sender address does not seem to be valid, and SPF \
                information does not grant $sender_host_address explicit \
                authority to send mail from $sender_address_domain
  log_message = SPF check failed.
  !verify     = sender/callout,random,postmaster
  !spf        = pass


# Add a SPF-Received: header to the message
warn
  message     = $spf_received
```

## A.7.2. SPF checks via Mail::SPF::Query

`Mail::SPF::Query` is a the official SPF test suite, available from http://spf.pobox.com/downloads.html. Debian users, install `libmail-spf-query-perl`.

The `Mail::SPF::Query` package comes with a daemon (**spfd**) that listens for requests on a UNIX domain socket. Unfortunately, it does not come with an "init" script to start this daemon automatically. Therefore, in the following example, we will use the standalone **spfquery** utility to make our SPF requests.

As above, insert the following prior to any greylisting checks and/or the final `accept` statement in acl_rcpt_to:

```
# Use "spfquery" to obtain SPF status for this particular sender/host.
# If the return code of that command is 1, this is an unauthorized sender.
#
deny
  message     = [SPF] $sender_host_address is not allowed to send mail \
                from $sender_address_domain.
  log_message = SPF check failed.
  set acl_m9  = -ipv4=$sender_host_address \
                -sender=$sender_address \
                -helo=$sender_helo_name
  set acl_m9  = ${run{/usr/bin/spfquery $acl_m9}}
  condition   = ${if eq {$runrc}{1}{true}{false}}
```

## A.8. Adding MIME and Filetype Checks

These checks depend on features found in Tom Kistner's `Exiscan−ACL` patch – see Prerequisites for details.

Exiscan−ACL includes support for MIME decoding, and file name suffix checks (or to use a misnomer from the Windows world, "file extension" checks). This check alone will block most Windows virii – but not those that are transmitted in `.ZIP` archives or those that exploit Outlook/MSIE HTML rendering vulnerabilities – see the discussion on Virus Scanners.

These checks should go into acl_data, before the final `accept` statement:

```
  # Reject messages that have serious MIME errors.
  #
  deny
    message     = Serious MIME defect detected ($demime_reason)
    demime      = *
    condition   = ${if >{$demime_errorlevel}{2}{1}{0}}


  # Unpack MIME containers and reject file extensions used by worms.
  # This calls the demime condition again, but it will return cached results.
  # Note that the extension list may be incomplete.
  #
  deny
    message     = We do not accept ".$found_extension" attachments here.
    demime      = bat:btm:cmd:com:cpl:dll:exe:lnk:msi:pif:prf:reg:scr:vbs:url
```

You will note that the `demime` condition is invoked twice in the example above. However, the results are cached, so the message is not actually processed twice.

## A.9. Adding Anti−Virus Software

Exiscan−ACL plugs into a number of different virus scanners directly, or any other scanner that can be run from the command line via its `cmdline` backend.

To use this feature, the main section of your Exim configuration file must specify which virus scanner to use, along with any options you wish to pass to that scanner. The basic syntax is:

```
av_scanner = scanner-type:option1:option:...
```

For instance:

```
av_scanner = sophie:/var/run/sophie
av_scanner = kavdaemon:/opt/AVP/AvpCtl
av_scanner = clamd:127.0.0.1 1234
av_scanner = clamd:/opt/clamd/socket
av_scanner = cmdline:/path/to/sweep −all −rec −archive %s:found:'(.+)'
...
```

In the DATA ACL, you then want to use the `malware` condition to perform the actual scanning:

```
  deny
```

```
    message  = This message contains a virus ($malware_name)
    demime   = *
    malware  = */defer_ok
```

The included file `exiscan-acl-spec.txt` contains full usage information.

# A.10. Adding SpamAssassin

Invoking SpamAssassin at SMTP–time is commonly done in either of two ways in Exim:

- Via the `spam` condition offered by `Exiscan-ACL`. This is the mechanism we will cover here.
- Via `SA-Exim`, another utility written by Marc Merlins (`<marc (at) merlins.org>`), specifically for running SpamAssassin at SMTP time in Exim. This program operates through Exim's `local_scan()` interface, either patched directly into the Exim source code, or via Marc's own `dlopen()` plugin (which, by the way, is included in Debian's `exim4-daemon-light` and `exim4-daemon-heavy` packages).

  `SA-Exim` offers some other features as well, namely *greylisting* and *teergrubing*. However, because the scan happens after the message data has been received, neither of these two features may be as useful as they would be earlier in the SMTP transaction.

  `SA-Exim` can be found at: http://marc.merlins.org/linux/exim/sa.html.

## A.10.1. Invoke SpamAssassin via Exiscan

`Exiscan-ACL`'s "`spam`" condition passes the message through either SpamAssassin or Brightmail, and triggers if these indicate that the message is junk. By default, it connects to a SpamAssassin daemon (`spamd`) running on `localhost`. The host address and port can be changed by adding a `spamd_address` setting in the *main* section of the Exim configuration file. For more information, see the `exiscan-acl-spect.txt` file included with the patch.

In our implementation, we are going to reject messages classified as spam. However, we would like to keep a copy of such messages in a separate mail folder, at least for the time being. This is so that the user can periodically scan for *False Positive*s.

Exim offers *controls* that can be applied to a message that is accepted, such as `freeze`. The Exiscan–ACL patch adds one more of these controls, namely `fakereject`. This causes the following SMTP response:

```
550-FAKEREJECT id=message-id
550-Your message has been rejected but is being kept for evaluation.
550 If it was a legit message, it may still be delivered to the target recipient(s).
```

We can incorporate this feature into our implementation, by inserting the following snippet in acl_data, prior to the final `accept` statement:

```
  # Invoke SpamAssassin to obtain $spam_score and $spam_report.
  # Depending on the classification, $acl_m9 is set to "ham" or "spam".
  #
  # If the message is classified as spam, pretend to reject it.
  #
  warn
```

```
    set acl_m9  = ham
    spam        = mail
    set acl_m9  = spam
    control     = fakereject
    logwrite    = :reject: Rejected spam (score $spam_score): $spam_report

  # Add an appropriate X-Spam-Status: header to the message.
  #
  warn
    message     = X-Spam-Status: \
                  ${if eq {$acl_m9}{spam}{Yes}{No}} (score $spam_score)\
                  ${if def:spam_report {: $spam_report}}
    logwrite    = :main: Classified as $acl_m9 (score $spam_score)
```

In this example, `$acl_m9` is initially set to "ham". Then SpamAssassin is invoked as the user `mail`. If the message is classified as spam, then `$acl_m9` is set to "spam", and the `FAKEREJECT` response above is issued. Finally, an `X-Spam-Status:` header is added to the message. The idea is that the *Mail Delivery Agent* or the recipient's *Mail User Agent* can use this header to filter junk mail into a separate folder.

## A.10.2. Configure SpamAssassin

By default, SpamAssassin presents its report in a verbose, table–like format, mainly suitable for inclusion in or attachment to the message body. In our case, we want a terse report, suitable for the `X-Spam-Status:` header in the example above. To do this, we add the following snippet in its site specific configuration file (`/etc/spamassassin/local.cf`, `/etc/mail/spamassassin/local.cf`, or similar):

```
### Report template
clear_report_template
report "_TESTSSCORES(, )_"
```

Also, a Bayesian scoring feature is built in, and is turned on by default. We normally want to turn this off, because it requires training that will be specific to each user, and thus is not suitable for system–wide SMTP time filtering:

```
### Disable Bayesian scoring
use_bayes 0
```

For these changes to take effect, you have to restart the SpamAssassin daemon (**spamd**).

## A.10.3. User Settings and Data

Say you have a number of users that want to specify their individual SpamAssassin preferences, such as the spam threshold, acceptable languages and character sets, white/blacklisted senders, and so on. Or perhaps they really want to be able to make use of SpamAssassin's native Bayesian scoring (though I don't see why[17]).

As discussed in the User Settings and Data section earlier in the document, there is a way for this to happen. We need to limit the number of recipients we accept per incoming mail delivery to one. We accept the first **RCPT TO:** command issued by the caller, then defer subsequent ones using a **451** SMTP response. As with greylisting, if the caller is a well–behaved MTA it will know how to interpret this response, and retry later.

### A.10.3.1. Tell Exim to accept only one recipient per delivery

In the acl_rcpt_to, we insert the following statement after validating the recipient address, but before any `accept` statements pertaining to unauthenticated deliveries from remote hosts to local users (i.e. before any greylist checks, envelope signature checks, etc):

```
# Limit the number of recipients in each incoming message to one
# to support per-user settings and data (e.g. for SpamAssassin).
#
# NOTE: Every mail sent to several users at your site will be
#       delayed for 30 minutes or more per recipient.  This
#       significantly slow down the pace of discussion threads
#       involving several internal and external parties.
#
defer
  message      = We only accept one recipient at a time - please try later.
  condition    = $recipients_count
```

### A.10.3.2. Pass the recipient username to SpamAssassin

In acl_data, we modify the `spam` condition given in the previous section, so that it passes on to SpamAssassin the username specified in the local part of the recipient address.

```
# Invoke SpamAssassin to obtain $spam_score and $spam_report.
# Depending on the classification, $acl_m9 is set to "ham" or "spam".
#
# We pass on the username specified in the recipient address,
# i.e. the portion before any '=' or '@' character, converted
# to lowercase.  Multiple recipients should not occur, since
# we previously limited delivery to one recipient at a time.
#
# If the message is classified as spam, pretend to reject it.
#
warn
  set acl_m9  = ham
  spam        = ${lc:${extract{1}{=@}{$recipients}{$value}{mail}}}
  set acl_m9  = spam
  control     = fakereject
  logwrite    = :reject: Rejected spam (score $spam_score): $spam_report
```

Note that instead of using Exim's `${local_part:...}` function to get the username, we manually extracted the portion before any "@" or "=" character. This is because we will use the latter character in our envelope signature scheme, to follow.

### A.10.3.3. Enable per–user settings in SpamAssassin

Let us now again look at SpamAssassin. First of all, you may choose to remove the `use_bayes 0` setting that we previously added in its site–wide configuration file. In any case, each user will now have the ability to decide whether to override this setting for themselves.

If mailboxes on your system map directly to local UNIX accounts with home directories, you are done. By default, the SpamAssassin daemon (**spamd**) performs a `setuid()` to the username we pass to it, and stores user data and settings in that user's home directory.

If this is not the case (for instance, if your mail accounts are managed by Cyrus SASL or by another server), you need to tell SpamAssassin where to find each user's preferences and data files. Also, **spamd** needs to keep running as a specific local user instead of attempting to `setuid()` to a non–existing user.

We do these things by specifying the options passed to **spamd** at startup:

- On a Debian system, edit the `OPTIONS=` setting in `/etc/default/spamassassin`.
- On a RedHat system, edit the `SPAMDOPTIONS=` setting in `/etc/sysconfig/spamassassin`.
- Others, figure it out.

The options you need are:

- `-u` *`username`* – specify the user under which **spamd** will run (e.g. `mail`)
- `-x` – disable configuration files in user's home directory.
- `--virtual-config-dir=/var/lib/spamassassin/%u` – specify where per–user settings and data are stored. "%u" is replaced with the calling username. **spamd** must be able to create or modify this directory:

```
# mkdir /var/lib/spamassassin
# chown -R mail:mail /var/lib/spamassassin
```

Needless to say, after making these changes, you need to restart **spamd**.

## A.11. Adding Envelope Sender Signatures

Here we implement Envelope Sender Signature in our outgoing mail, and check for these signatures before accepting incoming "bounces" (i.e. mail with no envelope sender).

The envelope sender address of outgoing mails from your host will be modified as follows:

```
sender=recipient=recipient.domain=hash@sender.domain
```

However, because this scheme may produce unintended consequences (e.g. in the case of mailing list servers), we make it optional for your users. We sign the envelope sender address of outgoing mail only if we find a file named ".return–path–sign" in the sender's home directory, and only if the domain we are sending to is matched in that file. If the file exists, but is empty, all domains match.

Similarly, we only require the recipient address to be signed in incoming "bounce" messages (i.e. messages with no envelope sender) if the same file exists in recipient's home directory. Users can exempt specific hosts from this check via their user specific whitelist, as described in Exempting Forwarded Mail.

Also, because this scheme involves tweaking with routers and transports in addition to ACLs, we do not include it in the Final ACLs to follow. If you are able to follow the instructions pertaining to those sections, you should also be able to add the ACL section as described here.

### A.11.1. Create a Transport to Sign the Sender Address

First we create an Exim *transport* that will be used to sign the envelope sender for remote deliveries:

```
remote_smtp_signed:
  debug_print    = "T: remote_smtp_signed for $local_part@$domain"
```

```
  driver         = smtp
  max_rcpt       = 1
  return_path    = $sender_address_local_part=$local_part=$domain=\
                   ${hash_8:${hmac{md5}{SECRET}{${lc:\
                     $sender_address_local_part=$local_part=$domain}}}}\
                   @$sender_address_domain
```

The "local part" of the sender address now consists of the following components, separated by equal signs ("="):

- the sender's username, i.e. the original local part,
- the local part of the recipient address,
- the domain part of the recipient address,
- a string unique to this sender/recipient combination, generated by:
    - encrypting the three prior components of the rewritten sender address, using Exim's `${hmac{md5}...}` function along with the `SECRET` we declared in the `main` section, [18]
    - hashing the result into 8 lowercase letters, using Exim's `${hash...}` function.

If you need authentication for deliveries to "smarthosts", add an appropriate `hosts_try_auth` line here as well. (Take it from your existing smarthost transport).

## A.11.2. Create a New Router for Remote Deliveries

Add a new router prior to the existing router(s) that currently handles your outgoing mail. This router will use the transport above for remote deliveries, but only if the file ".return−path−sign" exists in the sender's home directory, and if the recipient's domain is matched in that file. For instance, if you send mail directly over the internet to the final destination:

```
# Sign the envelope sender address (return path) for deliveries to
# remote domains if the sender's home directory contains the file
# ".return-path-sign", and if the remote domain is matched in that
# file. If the file exists, but is empty, the envelope sender
# address is always signed.
#
dnslookup_signed:
  debug_print   = "R: dnslookup_signed for $local_part@$domain"
  driver        = dnslookup
  transport     = remote_smtp_signed
  senders       = ! : *
  domains       = ! +local_domains : !+relay_to_domains : \
             ${if exists {/home/$sender_address_local_part/.return-path-sign}\
                         {/home/$sender_address_local_part/.return-path-sign}\
                         {!*}}
  no_more
```

Or if you use a smarthost:

```
# Sign the envelope sender address (return path) for deliveries to
# remote domains if the sender's home directory contains the file
# ".return-path-sign", and if the remote domain is matched in that
# file. If the file exists, but is empty, the envelope sender
# address is always signed.
#
```

```
smarthost_signed:
  debug_print   = "R: smarthost_signed for $local_part@$domain"
  driver        = manualroute
  transport     = remote_smtp_signed
  senders       = ! : *
  route_list    = * smarthost.address
  host_find_failed = defer
  domains       = ! +local_domains : !+relay_to_domains : \
             ${if exists {/home/$sender_address_local_part/.return-path-sign}\
                        {/home/$sender_address_local_part/.return-path-sign}\
                        {!*}}
  no_more
```

Add other options as you see fit (e.g. `same_domain_copy_routing = yes`), perhaps modelled after your existing routers.

Note that we do not use this router for mails with no envelope sender address – we wouldn't want to tamper with those! [19]

## A.11.3. Create New Redirect Router for Local Deliveries

Next, you need to tell Exim that incoming recipient addresses that match the format above should be delivered to the mailbox identified by the portion before the first equal ("=") sign. For this purpose, you want to insert a `redirect` router early in the `routers` section of your configuration file – before any other routers pertaining to local deliveries (such as a *system alias* router):

```
hashed_local:
  debug_print       = "R: hashed_local for $local_part@$domain"
  driver            = redirect
  domains           = +local_domains
  local_part_suffix = =*
  data              = $local_part@$domain
```

Recipient addresses that contain a equal sign are rewritten such that the portion of the local part that follows the equal sign are stripped off. Then all routers are processed again.

## A.11.4. ACL Signature Check

The final part of this scheme is to tell Exim that mails delivered to valid recipient addresses with this signature should *always* be accepted, and that other messages with a NULL envelope sender should be rejected if the recipient has opted in to this scheme. No greylisting should be done in either case.

The following snippet should be placed in acl_rcpt_to, prior to any SPF checks, greylisting, and/or the final `accept` statement:

```
  # Accept the recipient addresss if it contains our own signature.
  # This means this is a response (DSN, sender callout verification...)
  # to a message that was previously sent from here.
  #
  accept
    domains     = +local_domains
    condition   = ${if and {{match{${lc:$local_part}}{^(.*)=(.*)}}\
                            {eq{${hash_8:${hmac{md5}{SECRET}{$1}}}}{$2}}}\
                           {true}{false}}
```

```
# Otherwise, if this message claims to be a bounce (i.e. if there
# is no envelope sender), but if the receiver has elected to use
# and check against envelope sender signatures, reject it.
#
deny
  message     = This address does not match a valid, signed \
                return path from here.\n\
                You are responding to a forged sender address.
  log_message = bogus bounce.
  senders     = : postmaster@*
  domains     = +local_domains
  set acl_m9  = /home/${extract{1}{=}{${lc:$local_part}}}/.return-path-sign
  condition   = ${if exists {$acl_m9}{true}}
```

You will have an issue when sending mail to hosts that perform callout verification on addresses in the message *header*, such as the one provided in the `From:` field of your outgoing mail. The `deny` statement here will effectively give a negative response to such a verification attempt.

For that reason, you may want to convert the last `deny` statement into a `warn` statement, store the rejection message in `$acl_m0`, and perform the actual rejection after the **DATA** command, in a fashion similar to previously described:

```
# Otherwise, if this message claims to be a bounce (i.e. if there
# is no envelope sender), but if the receiver has elected to use
# and check against envelope sender signatures, store a reject
# message in $acl_m0, and a log message in $acl_m1.  We will later
# use these to reject the mail.  In the mean time, their presence
# indicate that we should keep stalling the sender.
#
warn
  senders     = : postmaster@*
  domains     = +local_domains
  set acl_m9  = /home/${extract{1}{=}{${lc:$local_part}}}/.return-path-sign
  condition   = ${if exists {$acl_m9}{true}}
  set acl_m0  = The recipient address <$local_part@$domain> does not \
                match a valid, signed return path from here.\n\
                You are responding to a forged sender address.
  set acl_m1  = bogus bounce for <$local_part@$domain>.
```

Also, even if the recipient has chosen to use envelope sender signatures in their outgoing mail, they may want to exempt specific hosts from having to provide this signature in incoming mail, even if the mail has no envelope sender address. This may be required for specific mailing list servers, see the discussion on Envelope Sender Signature for details.

## A.12. Accept Bounces Only for Real Users

As discussed in Accept Bounces Only for Real Users, there is now a loophole that prevents us from catching bogus *Delivery Status Notification* sent to system users and aliases, such as `postmaster`. Here we cover two alternate ways to ensure that bounces are only accepted for users that actually send outgoing mail.

## A.12.1. Check for Recipient Mailbox

The first method is performed in the acl_rcpt_to ACL. Here, we check that the recipient address corresponds to a local mailbox:

```
# Deny mail for users that do not have a mailbox (i.e. postmaster,
# webmaster...) if no sender address is provided.  These users do
# not send outgoing mail, so they should not receive returned mail.
#
deny
  message     = This address never sends outgoing mail. \
                You are responding to a forged sender address.
  log_message = bogus bounce for system user <$local_part@$domain>
  senders     = : postmaster@*
  domains     = +local_domains
  !mailbox check
```

Unfortunately, how we perform the `mailbox check` will depend on how you deliver your mail (as before, we extract the portion before the first "=" sign of the recipient address, to accomodate for Envelope Sender Signatures):

- If mailboxes map to local user accounts on your server, we can check that the recipient name maps to a user ID that corresponds to "regular" users on your system, e.g. in the range 500 – 60000:

  ```
  set acl_m9 = ${extract{1}{=}{${lc:$local_part}}}
  set acl_m9 = ${extract{2}{:}{${lookup passwd {$acl_m9}{$value}}}{0}}
  condition  = ${if and {{>={$acl_m9}{500}} {<${acl_m9}{60000}}} {true}}
  ```

- If you deliver mail to the Cyrus IMAP suite, you can use the provided **mbpath** command–line utility to check that the mailbox exists. You will want to make sure that the Exim user has permission to check for mailboxes (for instance, you may add it to the `cyrus` group: **# adduser exim4 cyrus**).

  ```
  set acl_m9 = ${extract{1}{=}{${lc:$local_part}}}
  condition  = ${run {/usr/sbin/mbpath -q -s user.$acl_m9} {true}}
  ```

- If you forward all mail to a remote machine for delivery, you may need to perform a Recipient Callout Verification and let that machine decide whether to accept the mail. You need to keep the original envelope sender intact in the callout:

  ```
  verify = recipient/callout=use_sender
  ```

Since in the case of locally delivered mail, this mailbox check duplicates some of the logic that is performed in the routers, and since it is specific to the mail delivery mechanism on our site, it is perhaps a bit kludgy for the perfectionists among us. So we will now provide an alternate way.

## A.12.2. Check for Empty Sender in Aliases Router

You probably have a router named `system_aliases` or similar, to redirect mail for users such as `postmaster` and `mailer-demon`. Typically, these aliases are not used in the sender address of outgoing mail. As such, you can ensure that incoming *Delivery Status Notification*s are not routed through it by adding the following condition to the router:

```
!senders = : postmaster@*
```

A sample aliases router may now look like this:

```
system_aliases:
  driver         = redirect
  domains        = +local_domains
  !senders       = : postmaster@*
  allow_fail
  allow_defer
  data           = ${lookup{$local_part}lsearch{/etc/aliases}}
  user           = mail
  group          = mail
  file_transport = address_file
  pipe_transport = address_pipe
```

Although we now block bounces to *some* system aliases, other aliases were merely shadowing existing system users (such as "root", "daemon", etc). If you deliver local mail through the the `accept` driver, and use `check_local_user` to validate the recipient address, you may now find yourself routing mail directly to these system accounts.

To fix this problem, we now want to add an additional condition in the router that handles your local mail (e.g. *local_user*) to ensure that the recipient not only exists, but is a "regular" user. For instance, as above, we can check that the user ID is in the range 500 – 60000:

```
  condition  = ${if and {{>={$local_user_uid}{500}}\
                         {<{$local_user_uid}{60000}}}\
                    {true}}
```

A sample router for local delivery may now look like this:

```
local_user:
  driver           = accept
  domains          = +local_domains
  check_local_user
  condition        = ${if and {{>={$local_user_uid}{500}}\
                               {<{$local_user_uid}{60000}}}\
                              {true}}
  transport        = transport
```

Beware that if you implement this method, the reject response from your server in response to bogus bounce mail for system users will be the same as for unknown recipients (**550 Unknown User** in our case).

## A.13. Exempting Forwarded Mail

After adding all these checks in the SMTP transaction, we may find ourselves indirectly creating collateral spam as a result of rejecting mails forwarded from trusted sources, such as mailing lists and mail accounts on other sites (see the discussion on Forwarded Mail for details). We now need to whitelist these hosts in order to exempt them from SMTP rejections –– at least those rejections that are caused by our spam and/or virus filtering.

In this example, we will consult two files in response to each **RCPT TO:** command:

- A global whitelist in `/etc/mail/whitelist-hosts`, containing backup MX hosts and other whitelisted senders [9], and
- A user–specific list in `/home/`*`user`*`/.forwarders`, specifying hosts from which that particuar user will receive forwarded mail (e.g. mailing list servers, outgoing mail servers for accounts

elsewhere...)

If your mail users do not have local user accounts and home directories, you may want to modify the file paths and/or lookup mechanisms to something more suitable for your system (e.g. database lookups or LDAP queries).

If the sender host is found in one of these whitelists, we save the word "accept" in `$acl_m0`, and clear the contents of `$acl_m1`, as described in the previous section on Selective Delays. This will indicate that we should not reject the mail in subsequent statements.

In the acl_rcpt_to, we insert the following statement after validating the recipient address, but before any `accept` statements pertaining to unauthenticated deliveries from remote hosts to local users (i.e. before any greylist checks, envelope signature checks, etc):

```
# Accept the mail if the sending host is matched in the global
# whitelist file.  Temporarily set $acl_m9 to point to this  file.
# If the host is found, set a flag in $acl_m0 and clear $acl_m1 to
# indicate that we should not reject this mail later.
#
accept
  set acl_m9  = /etc/mail/whitelist-hosts
  hosts       = ${if exists {$acl_m9}{$acl_m9}}
  set acl_m0  = accept
  set acl_m1  =


# Accept the mail if the sending host is matched in the ".forwarders"
# file in the recipient's home directory.  Temporarily set $acl_m9 to
# point to this file.  If the host is found, set a flag in $acl_m0 and
# clear $acl_m1 to indicate that we should not reject this mail later.
#
accept
  domains     = +local_domains
  set acl_m9  = /home/${extract{1}{=}{${lc:$local_part}}}/.forwarders
  hosts       = ${if exists {$acl_m9}{$acl_m9}}
  set acl_m0  = accept
  set acl_m1  =
```

In various statements in the acl_data ACL, we check the contents of `$acl_m0` to avoid rejecting the mail if this is set as per above. For instance, to avoid rejecting mail from whitelisted hosts due to a missing RFC2822 header:

```
deny
  message     = Your message does not conform to RFC2822 standard
  log_message = missing header lines
  !hosts      = +relay_from_hosts
  !senders    = : postmaster@*
  condition   = ${if !eq {$acl_m0}{accept}{true}}
  condition   = ${if or {{!def:h_Message-ID:}\
                         {!def:h_Date:}\
                         {!def:h_Subject:}} {true}{false}}
```

The appropriate checks are embedded in the Final ACLs, next.

---

# A.14. Final ACLs

OK, time to wake up! This has been very long reading – but congratulations on making it this far!

The following ACLs incorporate all of the checks we have described so in this implementation. However, some have been commented out, for the following reasons:

- Greylisting. This either requires additional software to be installed, or fairly complex inline configuration by way of additional ACLs and definitions in the Exim configuration file. I highly recommend it, though.
- Virus scanning. There is no *ubiquitous* scanner that nearly everyone uses, similar to SpamAssassin for spam identification. On the other hand, the documentation that comes with `Exiscan–ACL` should be easy to follow.
- Per–user settings for SpamAssassin. This is a trade–off that for many is unacceptable, as it involves deferring mail to all but the first recipient of a message.
- Envelope Sender Signatures. There are consequences, e.g. for roaming users. Also, it involves configuring routers and transports as well as ACLs. See that section for details.
- Accepting Bounces Only for Real Users. There are several ways of doing this, and determining which users are real is specific to how mail is delivered.

Without further ado, here comes the final result we have all been waiting for.

---

## A.14.1. acl_connect

```
# This access control list is used at the start of an incoming
# connection.  The tests are run in order until the connection is
# either accepted or denied.

acl_connect:
  # Record the current timestamp, in order to calculate elapsed time
  # for subsequent delays
  warn
    set acl_m2  = $tod_epoch


  # Accept mail received over local SMTP (i.e. not over TCP/IP).  We do
  # this by testing for an empty sending host field.
  # Also accept mails received over a local interface, and from hosts
  # for which we relay mail.
  accept
    hosts       = : +relay_from_hosts


  # If the connecting host is in one of several DNSbl's, then prepare
  # a warning message in $acl_c1.  We will later add this message to
  # the mail header.  In the mean time, its presence indicates that
  # we should keep stalling the sender.
  #

  warn
    !hosts      = ${if exists {/etc/mail/whitelist-hosts} \
                              {/etc/mail/whitelist-hosts}}
    dnslists    = list.dsbl.org : \
                  dnsbl.sorbs.net : \
                  dnsbl.njabl.org : \
```

```
                  bl.spamcop.net : \
                  dsn.rfc-ignorant.org : \
                  sbl-xbl.spamhaus.org : \
                  l1.spews.dnsbl.sorbs.net
    set acl_c1  = X-DNSbl-Warning: \
                  $sender_host_address is listed in $dnslist_domain\
                  ${if def:dnslist_text { ($dnslist_text)}}


  # Likewise, if reverse DNS lookup of the sender's host fails (i.e.
  # there is no rDNS entry, or a forward lookup of the resulting name
  # does not match the original IP address), then generate a warning
  # message in $acl_c1.  We will later add this message to the mail
  # header.
  warn
    condition   = ${if !def:acl_c1 {true}{false}}
    !verify     = reverse_host_lookup
    set acl_m9  = Reverse DNS lookup failed for host $sender_host_address
    set acl_c1  = X-DNS-Warning: $acl_m9


  # Accept the connection, but if we previously generated a message in
  # $acl_c1, stall the sender until 20 seconds has elapsed.
  accept
    set acl_m2  = ${if def:acl_c1 {${eval:20 + $acl_m2 - $tod_epoch}}{0}}
    delay       = ${if >{$acl_m2}{0}{$acl_m2}{0}}s
```

## A.14.2. acl_helo

```
# This access control list is used for the HELO or EHLO command in
# an incoming SMTP transaction. The tests are run in order until the
# greeting is either accepted or denied.

acl_helo:
  # Record the current timestamp, in order to calculate elapsed time
  # for subsequent delays
  warn
    set acl_m2  = $tod_epoch


  # Accept mail received over local SMTP (i.e. not over TCP/IP).
  # We do this by testing for an empty sending host field.
  # Also accept mails received from hosts for which we relay mail.
  #
  accept
    hosts       = : +relay_from_hosts


  # If the remote host greets with an IP address, then prepare a reject
  # message in $acl_c0, and a log message in $acl_c1.  We will later use
  # these in a "deny" statement.  In the mean time, their presence indicate
  # that we should keep stalling the sender.
  #
  warn
    condition   = ${if isip {$sender_helo_name}{true}{false}}
    set acl_c0  = Message was delivered by ratware
    set acl_c1  = remote host used IP address in HELO/EHLO greeting


  # Likewise if the peer greets with one of our own names
  #
```

```
  warn
    condition   = ${if match_domain{$sender_helo_name}\
                       {$primary_hostname:+local_domains:+relay_to_domains}\
                       {true}{false}}
    set acl_c0  = Message was delivered by ratware
    set acl_c1  = remote host used our name in HELO/EHLO greeting.


  # If HELO verification fails, we prepare a warning message in acl_c1.
  # We will later add this message to the mail header.  In the mean time,
  # its presence indicates that we should keep stalling the sender.
  #
  warn
    condition   = ${if !def:acl_c1 {true}{false}}
    !verify     = helo
    set acl_c1  = X-HELO-Warning: Remote host $sender_host_address \
                  ${if def:sender_host_name {($sender_host_name) }}\
                  incorrectly presented itself as $sender_helo_name
    log_message = remote host presented unverifiable HELO/EHLO greeting.


  # Accept the greeting, but if we previously generated a message in
  # $acl_c1, stall the sender until 20 seconds has elapsed.
  accept
    set acl_m2  = ${if def:acl_c1 {${eval:20 + $acl_m2 - $tod_epoch}}{0}}
    delay       = ${if >{$acl_m2}{0}{$acl_m2}{0}}s
```

## A.14.3. acl_mail_from

```
# This access control list is used for the MAIL FROM: command in an
# incoming SMTP transaction.  The tests are run in order until the
# sender address is either accepted or denied.
#

acl_mail_from:
  # Record the current timestamp, in order to calculate elapsed time
  # for subsequent delays
  warn
    set acl_m2  = $tod_epoch


  # Accept mail received over local SMTP (i.e. not over TCP/IP).
  # We do this by testing for an empty sending host field.
  # Also accept mails received from hosts for which we relay mail.
  #
  # Sender verification is omitted here, because in many cases
  # the clients are dumb MUAs that don't cope well with SMTP
  # error responses.
  #
  accept
    hosts     = : +relay_from_hosts


  # Accept if the message arrived over an authenticated connection,
  # from any host. Again, these messages are usually from MUAs.
  #
  accept
    authenticated = *
```

```
  # If present, the ACL variables $acl_c0 and $acl_c1 contain rejection
  # and/or warning messages to be applied to every delivery attempt in
  # in this SMTP transaction.  Assign these to the corresponding
  # $acl_m{0,1} message-specific variables, and add any warning message
  # from $acl_m1 to the message header.  (In the case of a rejection,
  # $acl_m1 actually contains a log message instead, but this does not
  # matter, as we will discard the header along with the message).
  #
  warn
    set acl_m0  = $acl_c0
    set acl_m1  = $acl_c1
    message     = $acl_c1


  # If sender did not provide a HELO/EHLO greeting, then prepare a reject
  # message in $acl_m0, and a log message in $acl_m1.  We will later use
  # these in a "deny" statement.  In the mean time, their presence indicate
  # that we should keep stalling the sender.
  #
  warn
    condition   = ${if def:sender_helo_name {0}{1}}
    set acl_m0  = Message was delivered by ratware
    set acl_m1  = remote host did not present HELO/EHLO greeting.


  # If we could not verify the sender address, create a warning message
  # in $acl_m1 and add it to the mail header.  The presence of this
  # message indicates that we should keep stalling the sender.
  #
  # You may choose to omit the "callout" option.  In particular, if
  # you are sending outgoing mail through a smarthost, it will not
  # give any useful information.
  #
  warn
    condition   = ${if !def:acl_m1 {true}{false}}
    !verify     = sender/callout
    set acl_m1  = Invalid sender <$sender_address>
    message     = X-Sender-Verify-Failed: $acl_m1
    log_message = $acl_m1


  # Accept the sender, but if we previously generated a message in
  # $acl_c1, stall the sender until 20 seconds has elapsed.
  accept
    set acl_m2  = ${if def:acl_c1 {${eval:20 + $acl_m2 - $tod_epoch}}{0}}
    delay       = ${if >{$acl_m2}{0}{$acl_m2}{0}}s
```

## A.14.4. acl_rcpt_to

```
# This access control list is used for every RCPT command in an
# incoming SMTP message.  The tests are run in order until the
# recipient address is either accepted or denied.

acl_rcpt_to:

  # Accept mail received over local SMTP (i.e. not over TCP/IP).
  # We do this by testing for an empty sending host field.
  # Also accept mails received from hosts for which we relay mail.
  #
  # Recipient verification is omitted here, because in many
```

```
# cases the clients are dumb MUAs that don't cope well with
# SMTP error responses.
#
accept
  hosts       = : +relay_from_hosts


# Accept if the message arrived over an authenticated connection,
# from any host. Again, these messages are usually from MUAs, so
# recipient verification is omitted.
#
accept
  authenticated = *


# Deny if the local part contains @ or % or / or | or !. These are
# rarely found in genuine local parts, but are often tried by people
# looking to circumvent relaying restrictions.
#
# Also deny if the local part starts with a dot. Empty components
# aren't strictly legal in RFC 2822, but Exim allows them because
# this is common.  However, actually starting with a dot may cause
# trouble if the local part is used as a file name (e.g. for a
# mailing list).
#
deny
  local_parts = ^.*[@%!/|] : ^\\.


# Deny if we have previously given a reason for doing so in $acl_m0.
# Also stall the sender for another 20s first.
#
deny
  message     = $acl_m0
  log_message = $acl_m1
  condition   = ${if and {{def:acl_m0}{def:acl_m1}} {true}}
  delay       = 20s


# If the recipient address is not in a domain for which we are handling
# mail, stall the sender and reject.
#
deny
  message     = relay not permitted
  !domains    = +local_domains : +relay_to_domains
  delay       = 20s


# If the address is in a local domain or in a domain for which are
# relaying, but is invalid, stall and reject.
#
deny
  message     = unknown user
  !verify     = recipient/callout=20s,defer_ok,use_sender
  delay       = ${if def:sender_address {1m}{0s}}



# Drop the connection if the envelope sender is empty, but there is
# more than one recipient address.  Legitimate DSNs are never sent
# to more than one address.
#
```

```
drop
  message      = Legitimate bounces are never sent to more than one \
                 recipient.
  senders      = : postmaster@*
  condition    = $recipients_count
  delay        = 5m


# --------------------------------------------------------------------
# Limit the number of recipients in each incoming message to one
# to support per-user settings and data (e.g. for SpamAssassin).
#
# NOTE: Every mail sent to several users at your site will be
#       delayed for 30 minutes or more per recipient.  This
#       significantly slow down the pace of discussion threads
#       involving several internal and external parties.
#       Thus, it is commented out by default.
#
#defer
#  message      = We only accept one recipient at a time - please try later.
#  condition    = $recipients_count
# --------------------------------------------------------------------


# Accept the mail if the sending host is matched in the ".forwarders"
# file in the recipient's home directory.  Temporarily set $acl_m9 to
# point to this file.  If the host is found, set a flag in $acl_m0 and
# clear $acl_m1 to indicate that we should not reject this mail later.
#
accept
  domains     = +local_domains
  set acl_m9  = /home/${extract{1}{=}{${lc:$local_part}}}/.forwarders
  hosts       = ${if exists {$acl_m9}{$acl_m9}}
  set acl_m0  = accept
  set acl_m1  =


# Accept the mail if the sending host is matched in the global
# whitelist file.  Temporarily set $acl_m9 to point to this  file.
# If the host is found, set a flag in $acl_m0 and clear $acl_m1 to
# indicate that we should not reject this mail later.
#
accept
  set acl_m9  = /etc/mail/whitelist-hosts
  hosts       = ${if exists {$acl_m9}{$acl_m9}}
  set acl_m0  = accept
  set acl_m1  =


# --------------------------------------------------------------------
# Envelope Sender Signature Check.
# This is commented out by default, because it requires additional
# configuration in the 'transports' and 'routers' sections.
#
# Accept the recipient addresss if it contains our own signature.
# This means this is a response (DSN, sender callout verification...)
# to a message that was previously sent from here.
#
#accept
#  domains     = +local_domains
#  condition   = ${if and {{match{${lc:$local_part}}{^(.*)=(.*)}}\
#                          {eq{${hash_8:${hmac{md5}{SECRET}{$1}}}}{$2}}}\
```

```
#                          {true}{false}}
#
# Otherwise, if this message claims to be a bounce (i.e. if there
# is no envelope sender), but if the receiver has elected to use
# and check against envelope sender signatures, reject it.
#
#deny
#  message     = This address does not match a valid, signed \
#                return path from here.\n\
#                You are responding to a forged sender address.
#  log_message = bogus bounce.
#  senders     = : postmaster@*
#  domains     = +local_domains
#  set acl_m9  = /home/${extract{1}{=}{${lc:$local_part}}}/.return-path-sign
#  condition   = ${if exists {$acl_m9}{true}}
# --------------------------------------------------------------------


# --------------------------------------------------------------------
# Deny mail for local users that do not have a mailbox (i.e. postmaster,
# webmaster...) if no sender address is provided.  These users do
# not send outgoing mail, so they should not receive returned mail.
#
# NOTE: This is commented out by default, because the condition is
#       specific to how local mail is delivered.  If you want to
#       enable this check, uncomment one and only one of the
#       conditions below.
#
#deny
#  message     = This address never sends outgoing mail. \
#                You are responding to a forged sender address.
#  log_message = bogus bounce for system user <$local_part@$domain>
#  senders     = : postmaster@*
#  domains     = +local_domains
#  set acl_m9  = ${extract{1}{=}{${lc:$local_part}}}
#
# --- Uncomment the following 2 lines if recipients have local accounts:
#  set acl_m9  = ${extract{2}{:}{${lookup passwd {$acl_m9}{$value}}}{0}}
#  !condition  = ${if and {{>={$acl_m9}{500}} {<${acl_m9}{60000}}} {true}}
#
# --- Uncomment the following line if you deliver mail to Cyrus:
#  condition  = ${run {/usr/sbin/mbpath -q -s user.$acl_m9} {true}}
# --------------------------------------------------------------------



# Query the SPF information for the sender address domain, if any,
# to see if the sending host is authorized to deliver its mail.
# If not, reject the mail.
#
deny
  message     = [SPF] $sender_host_address is not allowed to send mail \
                from $sender_address_domain
  log_message = SPF check failed.
  spf         = fail


# Add a SPF-Received: line to the message header
warn
  message     = $spf_received
```

```
  # --------------------------------------------------------------------
  # Check greylisting status for this particular peer/sender/recipient.
  # Before uncommenting this statement, you need to install "greylistd".
  # See:  http://packages.debian.org/unstable/main/greylistd
  #
  # Note that we do not greylist messages with NULL sender, because
  # sender callout verification would break (and we might not be able
  # to send mail to a host that performs callouts).
  #
  #defer
  #  message     = $sender_host_address is not yet authorized to deliver mail \
  #                from <$sender_address> to <$local_part@$domain>. \
  #                Please try later.
  #  log_message = greylisted.
  #  domains     = +local_domains : +relay_to_domains
  #  !senders    = : postmaster@*
  #  set acl_m9  = $sender_host_address $sender_address $local_part@$domain
  #  set acl_m9  = ${readsocket{/var/run/greylistd/socket}{$acl_m9}{5s}{}{}}
  #  condition   = ${if eq {$acl_m9}{grey}{true}{false}}
  #  delay       = 20s
  # --------------------------------------------------------------------

  # Accept the recipient.
  accept
```

### A.14.5. acl_data

```
# This access control list is used for message data received via
# SMTP.  The tests are run in order until the recipient address
# is either accepted or denied.

acl_data:
  # Log some header lines
  warn
    logwrite    = Subject: $h_Subject:


  # Add Message-ID if missing in messages received from our own hosts.
  warn
    condition   = ${if !def:h_Message-ID: {1}}
    hosts       = +relay_from_hosts
    message     = Message-ID: <E$message_id@$primary_hostname>


  # Accept mail received over local SMTP (i.e. not over TCP/IP).
  # We do this by testing for an empty sending host field.
  # Also accept mails received from hosts for which we relay mail.
  #
  accept
    hosts       = : +relay_from_hosts

  # Accept if the message arrived over an authenticated connection, from
  # any host.
  #
  accept
    authenticated = *


  # Deny if we have previously given a reason for doing so in $acl_m0.
  # Also stall the sender for another 20s first.
  #
```

```
deny
  message     = $acl_m0
  log_message = $acl_m1
  condition   = ${if and {{def:acl_m0}{def:acl_m1}} {true}{false}}
  delay       = 20s


# enforce a message-size limit
#
deny
  message     = Message size $message_size is larger than limit of \
                MESSAGE_SIZE_LIMIT
  condition   = ${if >{$message_size}{MESSAGE_SIZE_LIMIT}{yes}{no}}


# Deny unless the addresses in the header is syntactically correct.
#
deny
  message     = Your message does not conform to RFC2822 standard
  log_message = message header fail syntax check
  !verify     = header_syntax


# Uncomment the following to deny non-local messages without
# a Message-ID:, Date:, or Subject: header.
#
# Note that some specialized MTAs, such as certain mailing list
# servers, do not automatically generate a Message-ID for bounces.
# Thus, we add the check for a non-empty sender.
#
#deny
#  message     = Your message does not conform to RFC2822 standard
#  log_message = missing header lines
#  !hosts      = +relay_from_hosts
#  !senders    = : postmaster@*
#  condition   = ${if !eq {$acl_m0}{accept}{true}}
#  condition   = ${if or {{!def:h_Message-ID:}\
#                         {!def:h_Date:}\
#                         {!def:h_Subject:}} {true}{false}}


# Warn unless there is a verifiable sender address in at least
# one of the "Sender:", "Reply-To:", or "From:" header lines.
#
warn
  message     = X-Sender-Verify-Failed: No valid sender in message header
  log_message = No valid sender in message header
  !verify     = header_sender



# ----------------------------------------------------------------------
# Perform greylisting on messages with no envelope sender here.
# We did not subject these to greylisting after RCPT TO: because
# that would interfere with remote hosts doing sender callouts.
# Note that the sender address is empty, so we don't bother using it.
#
# Before uncommenting this statement, you need to install "greylistd".
# See:  http://packages.debian.org/unstable/main/greylistd
#
#defer
#  message     = $sender_host_address is not yet authorized to send \
```

```
#                delivery status reports to <$recipients>. \
#                Please try later.
#  log_message = greylisted.
#  senders     = : postmaster@*
#  condition   = ${if !eq {$acl_m0}{accept}{true}}
#  set acl_m9  = $sender_host_address $recipients
#  set acl_m9  = ${readsocket{/var/run/greylistd/socket}{$acl_m9}{5s}{}{}}
#  condition   = ${if eq {$acl_m9}{grey}{true}{false}}
#  delay       = 20s
# --------------------------------------------------------------------



# --- BEGIN EXISCAN configuration ---

# Reject messages that have serious MIME errors.
#
deny
  message     = Serious MIME defect detected ($demime_reason)
  demime      = *
  condition   = ${if >{$demime_errorlevel}{2}{1}{0}}


# Unpack MIME containers and reject file extensions used by worms.
# This calls the demime condition again, but it will return cached results.
# Note that the extension list may be incomplete.
#
deny
  message     = We do not accept ".$found_extension" attachments here.
  demime      = bat:btm:cmd:com:cpl:dll:exe:lnk:msi:pif:prf:reg:scr:vbs:url


# Messages larger than MESSAGE_SIZE_SPAM_MAX are accepted without
# spam or virus scanning
accept
  condition   = ${if >{$message_size}{MESSAGE_SIZE_SPAM_MAX} {true}}
  logwrite    = :main: Not classified \
                (message size larger than MESSAGE_SIZE_SPAM_MAX)


# --------------------------------------------------------------------
# Anti-Virus scanning
# This requires an 'av_scanner' setting in the main section.
#
#deny
#  message  = This message contains a virus ($malware_name)
#  demime   = *
#  malware  = */defer_ok
# --------------------------------------------------------------------



# Invoke SpamAssassin to obtain $spam_score and $spam_report.
# Depending on the classification, $acl_m9 is set to "ham" or "spam".
#
# If the message is classified as spam, and we have not previously
# set $acl_m0 to indicate that we want to accept it anyway, pretend
# reject it.
#
warn
  set acl_m9  = ham
  # ------------------------------------------------------------------
```

```
  # If you want to allow per-user settings for SpamAssassin,
  # uncomment the following line, and comment out "spam = mail".
  # We pass on the username specified in the recipient address,
  # i.e. the portion before any '=' or '@' character, converted
  # to lowercase.  Multiple recipients should not occur, since
  # we previously limited delivery to one recipient at a time.
  #
  # spam        = ${lc:${extract{1}{=@}{$recipients}{$value}{mail}}}
  # ------------------------------------------------------------------
  spam        = mail
  set acl_m9  = spam
  condition   = ${if !eq {$acl_m0}{accept}{true}}
  control     = fakereject
  logwrite    = :reject: Rejected spam (score $spam_score): $spam_report



# Add an appropriate X-Spam-Status: header to the message.
#
warn
  message     = X-Spam-Status: \
                ${if eq {$acl_m9}{spam}{Yes}{No}} (score $spam_score)\
                ${if def:spam_report {: $spam_report}}
  logwrite    = :main: Classified as $acl_m9 (score $spam_score)


# --- END EXISCAN configuration ---


# Accept the message.
#
accept
```

# Glossary

These are definitions for some of the words and terms that are used throughout this document.

## B

***Bayesian Filters***

A filter that assigns a probability of spam based on the recurrence of words (or, more recently, word constellations/phrases) between messages.

You initially train the filter by feeding it known junk mail (spam) and known legitimate mail (ham). A bayesian score is then be assigned to each word (or phrase) in each message, indicating whether this particular word or phrase occurs most commonly in ham or in spam. The word, along with its score, is stored in a *bayesian index*.

Such filters may catch indicators that may be missed by human programmers trying to manually create keyword−based filters. At the very least, they automate this task.

Bayesian word indexes are most certainly specific to the language in which they received training. Moreover, they are specific to individual users. Thus, they are perhaps more suitable for individual content filters (e.g. in *Mail User Agent*s) than they are for system−wide, SMTP−time filtering.

Moreover, spammers have developed techniques to defeat simple bayesian filters, by including random dictionary words and/or short stories in their messages. This decreases the spam probability assigned by a baynesian filter, and in the long run, degrades the quality of the bayesian index.

See also: http://www.everything2.com/index.pl?node=Bayesian.

## C

***Collateral Damage***

Blocking of a legitimate sender host due to an entry in a DNS blocklist.

Some blocklists (like SPEWS) routinely list the entire IP address space of an ISP if they feel the ISP is not responsive to abuse complaints, thereby affecting *all* its customers.

See also: *False Positive*

***Collateral Spam***

Automated messages sent in response to an original message (mostly spam or malware) where the sender address is forged. Typical examples of collateral spam include virus scan reports ("You have a virus") or other *Delivery Status Notification*s).

## D

***Domain Name System***

(*abbrev: DNS*) The de−facto standard for obtaining information about internet domain names. Examples of such information include IP addresses of its servers (so−called *A records*), the dedication of incoming mail exchangers (MX records), generic server information (SRV records), and miscellaneous text information (TXT records).

DNS is a hierarctical, distributed system; each domain name is associated with a set of one or more DNS servers that provide information about that domain – including delegation of name service for its subdomains.

For instance, the top–level domain "org" is operated by The Public Interest Registry; its DNS servers delegate queries for the domain name "tldp.org" to specific name servers for The Linux Documentation Project. In turn, TLDPs name server (actually operated by UNC) may or may not delegate queries for third–level names, such as "www.tldp.org".

DNS lookups are usually performed by forwarding name servers, such as those provided by an Internet Service Provider (e.g. via DHCP).

***Delivery Status Notification***

(*abbrev: DSN*) A message automatically created by an MTA or MDA, to inform the sender of an original messsage (usually included in the DSN) about its status. For instance, DSNs may inform the sender of the original message that it could not be delivered due to a temporary or permanent problem, and/or whether or not and for how long delivery attempts will continue.

Delivery Status Notifications are sent with an empty *Envelope Sender* address.

# E

***Envelope Sender***

The e–mail address given as sender of a message during the SMTP transaction, using the **MAIL FROM:** command. This may be different from the address provided in the "From:" header of the message itself.

One special case is *Delivery Status Notification* (bounced message, return receipt, vacation message..). For such mails, the *Envelope Sender* is empty. This is to prevent *Mail Loop*s, and generally to be able to distinguish these from "regular" mails.

See also: The SMTP Transaction

***Envelope Recipient***

The e–mail address(es) to which the message is sent. These are provided during the SMTP transaction, using the **RCPT TO** command. These may be different from the addresses provided in the "To:" and "Cc:" headers of the message itself.

See also: The SMTP Transaction

# F

***False Negative***

Junk mail (spam, virus, malware) that is misclassified as legitimate mail (and consequently, not filtered out).

***False Positive***

Legitimate mail that is misclassified as junk (and consequently, blocked).

See also: *Collateral Damage*.

***Fully Qualified Domain Name***
(a.k.a. "FQDN"). A full, globally unique, internet name, including DNS domain. For instance: "www.yahoo.com".

A *FQDN* does not always point to a single host. For instance, common service names such as "www" often point to many IP addresses, in order to provide some load balancing on the servers. However, the *primary* host name of a given machine should always be unique to that machine; for instance: "p16.www.scd.yahoo.com".

A *FQDN* always contains a period ("."). The part before the first period is the *unqualified name*, and is not globally unique.

# J

***Joe Job***
A spam designed to look like it came from someone else's valid address, often in a malicous attempt at generating complaints from third parties and/or cause other damage to the owner of that address.

See also: http://www.everything2.com/index.pl?node=Joe%20Job

# M

***Mail Delivery Agent***
(*abbrev: MDA*) Software that runs on the machine where a users' mailbox is located, to deliver mail into that mailbox. Often, that delivery is performed directly by the MTA *Mail Transport Agent*, which then serves a secondary role as an MDA. Examples of separate Mail Delivery Agents include: Deliver, Procmail, Cyrmaster and/or Cyrdeliver (from the Cyrus IMAP suite).

***Mail Loop***
A situation where one automated message triggers another, which directly or indirectly triggers the first message over again, and so on.

Imagine a mailing list where one of the subscribers is the address of the list itself. This situation is often dealt with by the list server adding an "X–Loop:" line in the message header, and not processing mails that already have one.

Another equivalent term is *Ringing*.

***Mail Transport Agent***
(*abbrev: MTA*) Software that runs on a mail server, such as the mail exchanger(s) of a internet domain, to send mail to and receive mail from other hosts. Popular MTAs include: Sendmail, Postfix, Exim, Smail.

***Mail User Agent***
(*abbrev: MUA*; a.k.a. *Mail Reader*) User software to access, download, read, and send mail. Examples include Microsoft Outlook/Outlook Express, Apple Mail.app, Mozilla Thunderbird, Ximian Evolution.

***Mail Exchanger***
(*abbrev: MX*) A machine dedicated to (sending and/or) receiving mail for an internet domain.

The DNS zone information for a internet domain normally contains a list of *Fully Qualified Domain Names* that act as incoming mail exchangers for that domain. Each such listing is called an "MX record", and it also contains a number indicating its "priority" among several "MX records". The listing with the lowest number has the first priority, and is considered the "primary mail exchanger" for that domain.

***Micropayment Schemes***

(a.k.a. *sender pay* schemes). The sender of a message expends some machine resources to create a virtual *postage stamp* for each recipient of a message − usually by solving a mathematical challenge that requires a large number of memory read/write operations, but is relatively CPU speed insensitive. This stamp is then added to the headers of the message, and the recipient would validate the stamp through a much simpler decoding operation.

The idea is that because the message requires a postage stamp for every recipient address, spamming hundreds or thousands of users at once would be prohibitively "expensive".

Two such systems are:

◊ Camram
◊ Microsoft's Penny Black Project

# O

***Open Proxy***

A *proxy* which openly accepts TCP/IP connections from anywhere, and forwards them anywhere.

These are typically exploited by spammers and virii, who use them to conceal their own IP address, and/or to more effectively distribute transmission loads across several hosts and networks.

See also: *Zombie Host*

***Open Relay***

A *Relay* which openly accepts mail from anywhere, and forwards them to anywhere.

In the 1980s, virtually every public SMTP server was an *Open Relay*. Messages would often travel between multiple third−party machines before it reached the intended recipient. Now, legitimate mail are almost exclusively sent directly from an outgoing *Mail Transport Agent* on the sender's end to the incoming *Mail Exchanger*(s) for the recipient's domain.

Conversely, *Open Relay* servers that still exist on the internet are almost exclusively exploited by spammers to hide their own identity, and to perform some load balancing on the task of sending out millions of messages, presumably before DNS blocklists have a chance to get all of these machines listed.

See also the discussion on Open Relay Prevention.

# P

***proxy***

A machine that acts on behalf of someone else. It may forward e.g. HTTP requests or TCP/IP connections, usually to or from the internet. For instance, companies − or sometimes entire countries − often use "Web Proxy Servers" to filter outgoing HTTP requests from their internal network. This may or may not be transparent to the end user.

See also: *Open Proxy*, *Relay*.

# R

***Ratware***

Mass−mailing virii and e−mail software used by spammers, specifically designed to deliver large amounts of mail in a very short time.

Most ratware implementations incorporate only as much SMTP client code as strictly neccessary to deliver mail in the best−case scenario. They provide false or inaccurate information in the SMTP dialogue with the receiving host. They do not wait for responses from the receiver before issuing commands, and disconnect if no response has been received in a few seconds. They do not follow normal retry mechanisms in case of temporary failures.

***Relay***

A machine that forwards e−mail, usually to or from the internet. One example of a relay is the "smarthost" that an ISP provides to its customers for sending outgoing mail.

See also: *Open Relay*, *proxy*

***Request for Comments***

(*abbrev: RFC*) From http://www.rfc−editor.org/: " The Request for Comments (RFC) document series is a set of technical and organizational notes about the internet [...]. Memos in the RFC series discuss many aspects of computer networking, incluing protocols, procedures, programs, and concepts, as well as meeting notes, opinions, and sometimes humor. "

These documents make up the "rules" internet conduct, including descriptions of protocols and data formats. Of particular interest for mail deliveries are:

◊ RFC 2821, "Simple Mail transfer Protocol", and
◊ RFC 2822, "Internet Message Format".

# S

***Spam Trap***

An e−mail address that is *seeded* to address−harvesting robots via public locations, then used to feed collaborative tools such as DNS Blacklists and Junk Mail Signature Repository.

Mails sent to these addresses are normally spam or malware. However, some of it will be *collateral*, spam − i.e. *Delivery Status Notification* to faked sender addresses. Thus, unless the spam trap has safeguards in place to disregard such messages, the resulting tool may not be completely reliable.

# Z

***Zombie Host***

A machine with an internet connection that is infected by a mass−mailing virus or worm. Such machines invariably run a flavor of the Microsoft® Windows® operating system, and are almost always in "residential" IP address blocks. Their owners either do not know or do not care that the machines are infected, and often, their ISP will not take any actions to shut them down.

Fortunately, there are various DNS blocklists, such as "dul.dnsbl.sorbs.net", that incorporate such "residential" address blocks. You should be able to use these blocklists to reject incoming mail. Legitimate mail from residential users should normally go through their ISP's "smarthost".

# Appendix B. GNU General Public License

## B.1. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software – to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

---

## B.2. TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

### B.2.1. Section 0

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program " means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification ".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

## B.2.2. Section 1

You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

## B.2.3. Section 2

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
2. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
3. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License.

**Exception:**

If the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

## B.2.4. Section 3

You may copy and distribute the Program (or a work based on it, under Section 2 in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

1. Accompany it with the complete corresponding machine−readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
2. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine−readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
3. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

## B.2.5. Section 4

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

## B.2.6. Section 5

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the

Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

### B.2.7. Section 6

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

### B.2.8. Section 7

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty−free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

### B.2.9. Section 8

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

### B.2.10. Section 9

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

### B.2.11. Section 10

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

### B.2.12. NO WARRANTY Section 11

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

### B.2.13. Section 12

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

## B.3. How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111−1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse−clicks or menu items−−whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

### Notes

[1] Untrusted third party hosts may still generate collateral spam if you reject the mail. However, unless that host is an *Open Proxy* or *Open Relay*, it presumably delivers mail only from legitimate senders, whose addresses are valid. If it *is* an Open Proxy or SMTP Relay − well, it is better that you reject the mail and let it freeze in *their* outgoing mail queue than letting it freeze in yours. Eventually, this ought to give the owners of that server a clue.

[2] Personally I do not think challenge/response systems are a good idea in any case. They generate *Collateral Spam*, they require special attention for mail sent from automated sources such as monthly bank statements, and they degrade the usability of e−mail as people need to jump through hoops to get in touch with each other. Many times, senders of legitimate mail will not bother to or know that they

need to follow up to the confirmation request, and the mail is lost.

[3] My view stands in sharp contrast to that of a large number of "spam hacktivists", such as the maintainers of the SPEWS blacklist. One of the stated aims of this list is precisely to inflict *Collateral Damage* as a means of putting pressure on ISPs to react on abuse complaints. Listing complaints are typically met with knee−jerk responses such as "bother your ISP, not us", or "get another ISP".

Often, these are not viable options. Consider developing countries. For that matter, consider the fact that nearly everywhere, broadband providers are regulated monopolies. I believe that these attitudes illustrate the exact crux of the problem with trusting these groups.

Put plainly, there are much better and more accurate ways available to filter junk mail.

[4] Beware that while you are holding up an incoming SMTP delivery, you are also holding up a TCP socket on your server, as well as memory and other server resources. If your server is generally busy, imposing SMTP transaction delays will make you more vulnerable to Denial−of−Service attacks. A more "scalable" option may be to drop the connection once you have conclusive evidence that the sender is a ratware client.

[5] Similar lists exist for different purposes. For instance, "bondedsender.org" is a *DNS whitelist* (DNSwl), containing "trusted" IP addresses, whose owners have posted a financial bond that will be debited in the event that spam originates from that address. Other lists contain IP addresses in use by specific countries, specific ISPs, etc.

[6] For instance, the outgoing mail exchangers ("smart hosts") of the world's largest Internet Service Provider (ISP), comcast.net, is as of the time of this writing included in the SPEWS *Level 1* list. Not wholly undeserved from the viewpoint that Comcast needs to more effectively enforce their own AUP, but this listing does affect 30% of all US internet users, mostly "innocent" subscribers such as myself.

To make matters worse, information published in the SPEWS FAQ states: *The majority of the Level 1 list is made up of netblocks owned by the spammers or spam support operations themselves, with few or no other legitimate customers detected.* Technically, this information is accurate if (a) you consider Comcast a "spam support operation", and (b) pay attention to the word "other". Word parsing aside, this information is clearly misleading.

[7] Although this check is normally quite effective at weeding out junk, there are reports of buggy L−Soft listserv installations that greet with the plain IP address of the list server.

[8] A special case is the NULL envelope sender address (i.e. **MAIL FROM:** <>) used in *Delivery Status Notification*s and other automatically generated responses. This address should always be accepted.

[9] Although rare, some "legitimate" bulk mail senders, such as `groups.yahoo.com`, will not retry temporarily failed deliveries. Evan Harris has compiled a list of such senders, suitable for whitelisting purposes: http://cvs.puremagic.com/viewcvs/greylisting/schema/whitelist_ip.txt?view=markup.

[10] Large sites often use multiple servers to handle outgoing mail. For instance, one server or pool of servers may be used for immediate delivery. If the first delivery attempt fails, the mail is handed off to a fallback server which has been tuned for large queues. Hence, from such sites, the first two delivery attempts will fail.

[11] Some specialized MTAs, such as certain mailing list servers, do not automatically generate a `Message-ID:` header for "bounced" messages (*Delivery Status Notification*s). These messages are identified by an empty *Envelope Sender*.

[12] The IMAP protocol does not allow for NUL characters to be transmitted to the mail user agent, so the Cyrus developers decided that the easiest way to deal with mails containing it was to reject them.

[13] Why on earth the authors of anti−virus software are stupid enough to trust the sender address in an e−mail containing a virus is perhaps a topic for a closer psychoanalytic study.

[14] In particular, Exim is perhaps most popular among users of Debian GNU/Linux, as it is the default MTA in that distribution. If you use Debian ("Sarge" or later), you can obtain Exim+Exiscan−ACL by installing the `exim4−daemon−heavy` package:

```
# apt−get install exim4−daemon−heavy
```

[15] *Debian users:* The `exim4−config` package gives you a choice between splitting the Exim configuration into several small chunks distributed within subdirectories below `/etc/exim4/conf.d`, or to keep the entire configuration in a single file.

If you chose the former option (I recommend this!), you can keep your customization well separated from the stock configuration provided with the `exim4−config` package by creating new files within these subdirectories, rather than modifying the existing ones. For instance, you may create a file named `/etc/exim4/conf.d/acl/80_local−config_rcpt_to` to declare your own ACL for the **RCPT TO:** command (see below).

The Exim "init" script (`/etc/init.d/exim4`) will automatically consolidate all these files into a single large run−time configuration file next time you (re)start.

[16] Debian users: As of July 14th, 2004, the version of Exiscan−ACL that is included in the `exim4−daemon−heavy` package does not yet have support for SPF. In the mean time, you may choose the other SPF implementation; install `libmail−spf−query−perl`.

[17] Although it is true that Bayesian training is specific to each user, it should be noted that SpamAssassin's Bayesian classifier is, IMHO, not that stellar in any case. Especially I find this to be the case since spammers have learned to defeat such systems by seeding random dictionary words or stories in their mail (e.g. in the metadata of HTML messages).

[18] If you think this is an overkill, would I tend to agree on the surface. In previous versions of this document, I simply used `${hash_8:SECRET=....}` to generate the last component of the signature. However, with this it would be technically possible, with a bit of insight into Exim's `${hash...}` function and some samples of your outgoing mail sent to different recipients, to forge the signature. Matthew Byng−Maddic <`mbm (at) colondot.net`> notes: *What you're writing is a document that you expect many people to just copy. Given that, kerchoff's principle starts applying, and all of your secrecy should be in the key. If the key can be reversed out, as seems likely with a few return paths, then the spammer kan once again start emitting valid return−paths from that domain, and you're back to where you started. [...] Better, IMO, to have it being strong from the start.*

[19] In the examples above, the `senders` condition is actually redundant, since the file `/home//.return−path−sign` is not likely to exist. However, we make the condition explicit for clarity.