# Spam Handling in Heterogeneous Environments

Florin Bogdan MANOLACHE, Octavian RUSU, Palavi ADUSUMILLI

**Abstract**

*This paper presents an overview of spam avoiding techniques used at the Mellon College of Science, Carnegie Mellon University. Various criteria for the mail system are taken into account, such as client compatibility with different platforms, migration tools, performance and safety, scalability, ease of use. The research presented here shows that the best compromise for a heterogeneous dynamic environment is offered by conservative mail server filters coupled with open source client level anti-spam software.*

## 1 Introduction

A modern mail system has to fulfill a tough set of criteria:

- Easy and flexible configuration: easily manage security, privacy, mailboxes, mail accounts.
- User friendly: automatic show attachments.
- Large storage space for email: because of lack of native Windows support for ssh, email systems are often used for backup or as file storage.
- Spam resistant: anti-spam solutions should offer fast configuration/training and easy to spot false positives.
- Easy access from different locations with (almost) no re-configuration and minimum bandwidth requirements (e.g. modem access).
- Uniformity: mail clients having consistent look and feel on most platforms and operating systems.

It is hard to determine which of these requirements are the most important, and there is no solution that will satisfy all of them. So users have to make compromises, depending on their needs, and opt for a solution that suits their needs the best. However, the most annoying and costly problem for the system administrator is how to efficiently deal with spam.

The computing environment at Carnegie Mellon University is highly heterogeneous. Each college tends to recommend and support a different set of operating systems, resulting in an interesting mix of Windows, Linux, MacOS, and Solaris environments. The university servers handle email for about 20,000 accounts. To use the email system, Computing Services is offering a web interface and access to IMAP/SMTP accounts. The structure of the service is shown in Fig. 1. Most of the blocks are implemented by a pool of servers through suitable load balancing mechanisms.

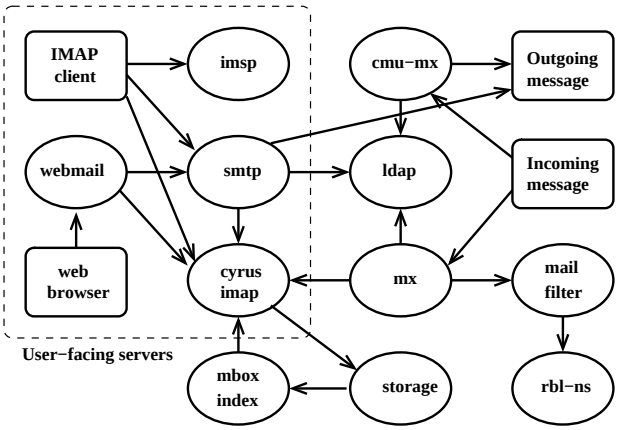


Figure 1: Mail system general structure.

Spam filtering can be done at the server level or at the client/mailreader level. In the next sections, server and client solutions will be described. Advantages of the two alternatives will be presented, and an attempt to find the best compromise according to the criteria presented above will be made.

## 2 Server level solutions

Server level filtering has several important advantages:

- independent of user-installed software;
- minimizes wasted bandwidth;
- easy to be managed and controlled by sysadmins;

and the following disadvantages:

- user level configuration can be implemented through scripting rules, which is beyond the knowledge level and patience of the typical user;

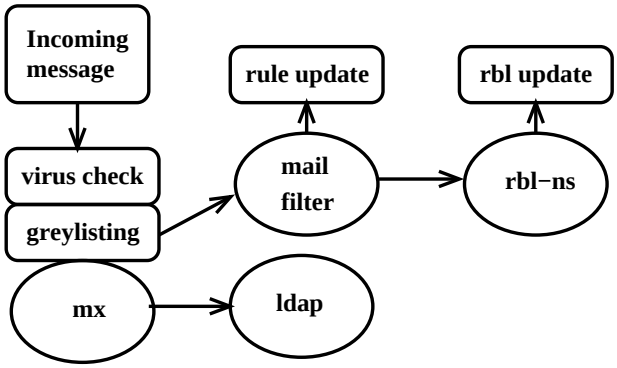


Figure 2: Mail system general structure.

- false positives can be hard to spot in a large amount of spam;
- server level spam filters are not very effective.

Typically, server level filters are implemented through a service like amavis that is plugged into a mail server like postfix or sendmail. A possible delivery scenario [1] starts with the server deferring the first attempt (many spambots won't retry), followed by a virus filter like clamav, and completed through a spam greylisting filter (Fig. 2). The greylisting filter compares the message against a set of rules and the IP address of the sender against a list of known spam relays. Based on the result, the message header gets a new field containing a spam score. In a conservative approach, only the messages with a very high spam score may be automatically deleted. Handling of messages with moderate or low spam scores is be done by the mail clients or client level filters.

The most important problem of the server level spam filters is the lack of real time adaptability. Spam is changing its shape quickly, new computers are compromised and transformed into spam relays, it is very hard for the configurations of these filters to keep up with the attacks. A solution is to subscribe to a commercial service that attempts to update the rules and RBL databases in real time. CMU is using PureMessageAS (Sophos) at this time. While the price is reasonably affordable, the effectiveness of the solution is lower than that of a free client level spam filter, as shown by the data presented in the next section.

The merits of using a conservatively configured server level anti-spam solution are the following:

- has virtually zero false positives;
- eliminates most of the spam, making easy for users to spot false positives introduced by client level filters;
- allows a decent experience from mail clients lacking advanced anti-spam features, as webmail interfaces and basic lightweight clients.

The next section analyzes some client level solutions, and presents a comparison of the effectiveness for the two alternatives on real world data.

## 3 Mail client effectiveness study

Various client level anti-spam solutions have highly different effectiveness. Each mail client may offer a different way of interfacing with spam filtering software, starting from scripting languages as procmail and sieve, to learning filters like the ones built into thunderbird/seamonkey clients. The only way of having efficient spam detection is continuous training of the filters.

The main advantages of filtering spam at client level are the following:

- empowers the users, lets them decide what is spam;
- much easier to configure interactively, resulting in more effective protection;

and the disadvantages are:

- false positives can be hard to spot in a large amount of spam;
- high probability of false positives;
- new training of the filters is needed on every computer/client.

Our first task was to pick the right software that would fulfill as many of the criteria presented in Section 1 as possible. The client with the best coverage of all the platforms was mozilla thunderbird. Thunderbird proved to be a reasonably stable software on most platform, user friendly, actively developed and maintained, easy to configure, with an intuitive GUI, and featuring a great spam filter that provided very good results after only a few days of training. After testing thunderbird for a few weeks, it was concluded that most of the built-in features built, like SSL, TLS, LDAP, were compatible and worked well with the CMU environment after some basic tweaking. One of the major inconveniences was the inability to run thunderbird remotely over modem connections. Another major complaint of the users was about migrating their old pine addressbook under thunderbird. Dealing with this problem is the subject of the next section.

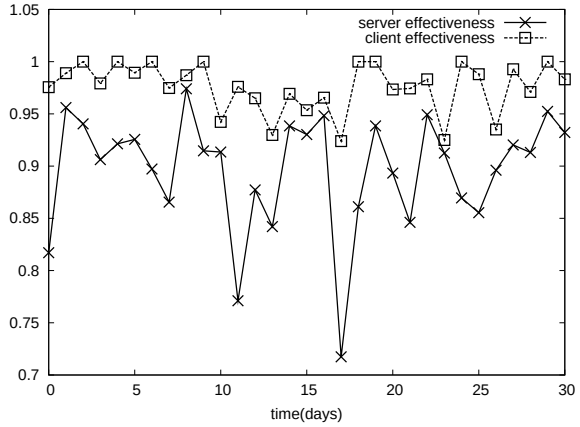


Figure 3: Spam detection effectiveness data.

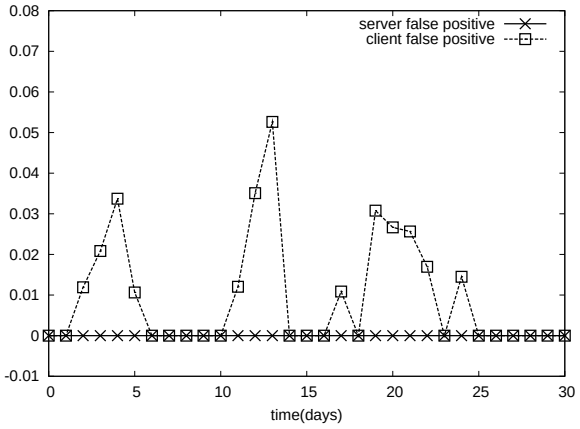


Figure 4: False positive data.

To justify a migration to thunderbird, a comparison of how efficient is its spam filter compared to the server level filter was necessary to be done. Several accounts with high exposure to spam were monitored for one month. The results are presented in Figs. 3,4. The percentage of spam detected by thunderbird is consistently better than the server level filter, with a typical improvement of 10-20% as shown in Fig. 3. This is an expected behavior, since the main objective of the server level filter is to eliminate as much spam as possible without false positives. Fig. 4 shows that indeed, only thunderbird filters generates false positives, although the percentage is very low (typically under 3% of the detected spam). The false positives are collected in the "spam" folder together with a low amount of spam that went undetected by the server side filters. That way, false positives can be easily spotted and used for further training of the thunderbird filter.

As this results suggest, a pure server-side filter solution is far from being efficient in the long run, because there is an important amount of undetected spam. On the other hand, a pure client filter would generate an important number of false positives that can easily slip in the large amount of spam. However, a combination of server and client level filters proved to offer a very efficient setup. That way, it is possible to have the advantages offered by both solutions.

# 4 Migration tools for Thunderbird

Surprisingly, the most time consuming part of maintaining the mail service is user migration. For historical reasons, the preferred mail reader at CMU for a very long time was pine. The tricky part to move an user from pine to thunderbird/seamonkey is to get the pine addressbook converted into a format that can be imported. Several websites exist on the Internet for this purpose ([2], [3], [4]). While some of offered tools do a decent job for individual addressbook entries, none of them handle distribution lists. So it was necessary to develop a set of scripts and interfaces for this purpose, presented in this section.

First, a perl script that can be used as a command line tool was written. That proved to be inconvenient for users accustomed mostly with a GUI. So we decided to give a try to the "software as a service" paradigm [5] and give access to our script through a web interface (http://ldif.math.cmu.edu/). While that forced us to seriously consider the security risks of running the script through the webserver, it resulted overall in a much better user experience for the transition. Another important advantage of the web interface was the ease to obtain feedback from the users and polish the script to work better.

## 4.1 Format of Pine Address Book

Pine address book has five fields separated by tabs: Nickname, Full name, Addresses, Fcc, Comment. Fcc specifies the folder to save sent mail. Addresses field contains E-mail addresses; if the entry is a single contact only one E-mail address is present. Distribution lists have email addresses or nicknames separated by commas and the list in enclosed in braces. If a field is not present, an empty tab is stored in the address book. If there are no fields after email address, a new line is present. If the field is longer, then the entry is split and second line starts with three spaces.

To import pine addressbook entries into thunderbird or seamonkey, the addressbook has first to be converted into LDIF format. LDIF is the only format that allows to import distribution lists. Other formats like CSV and VCARD cannot portably represent distribution lists in their data.

An Example addressbook contact is given below:

```
barni  Barney Hammer   barni@cmu.edu
abum   Bum, Arim       Arim Bum <arim@cs.cmu.edu>
   320 Dock Haven Line
tom     Leeman, Thoma  tleeman@math.cmu.edu
   Phone: 4555
bond    Bond, James    James Bond <bond@tec.edu>
```

### 4.2 Format of LDIF address book

LDIF format contains addressbook contacts separated by blank lines. Fields of the contact are separated by newlines. An Example LDIF contact is given below:

```
dn: cn=Florin Manolache,mail=florin@cmu.edu
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetOrgPerson
objectclass: mozillaAbPersonAlpha
givenName: Florin
sn: Manolache
cn: Florin Manolache
description: Wean Hall 2208, Ph No: 412-268-2000
mail: florin@cmu.edu
```

The dn field represents distinguished name which uniquely identifies the contact. The Display field in thunderbird address book is shown in LDIF format as cn field. The First field in thunderbird address book is shown in LDIF format as givenName. The Last field in thunderbird address book is shown in LDIF format as sn. The email address is shown in LDIF format as mail. The nick name field in thunderbird is exported as mozillaNicname The Notes field in thunderbird address book is exported as description field.

The object class fields determine whether the contact is an individual address or a distribution list. If the object class is person, then it is a contact; if it is groupOfNames, then it is a distribution list. Distribution lists have mail fields displayed in different lines by member fields, as shown below:

```
member: mail=padus@andrew.cmu.edu
member: mail=pal.a7@gmail.com
member: mail=pall@cmu.edu
```

The ldif format supports telephone numbers, addresses, title, and many other fields as well. For the purpose of pine to LDIF conversion, only required fields are investigated. For more details please refer to RFC 2849 [6].

### 4.3 Mapping of pine addressbook fields to LDIF

Pine addressbook contains the following entries: Nickname, Fullname, Addresses, Fcc, Comment. Nickname in pine is mapped to mozillaNicname in LDIF. Full name in pine is mapped to cn in LDIF. Addresses in pine are mapped to mail in ldif. Fcc in pine has no mapping in ldif, hence the field is ignored. Comment in pine is description in ldif.

### 4.4 Details of the script

**License:** The script is free software and can redistributed and/or modified under the terms of the GNU Lesser General Public License.

**Usage:** The file ab2ldif.pl is placed in the cgi-bin folder and is an executable Perl script; it takes as input a pine addressbook file through an html web interface.

**Input validation:** If the user tries to upload a binary file, the script responds with an "invalid file" message. Special shell characters like `><|\‘${}[]` in the input are discarded. To make a DoS attack more difficult, the script will parse up to a maximum number of records (currently set to 1000), and discard the rest. Each field in the record is limited to a maximum length (currently set to 50 characters). The size of the input file is limited (currently set to 1MB); if that limit is exceeded, the file will not be parsed and the script responds with a "limit exceeded" message.

**Parsing steps:**

- The script initially cleans up the address book by merging multi-line entries into single lines.
- Line by line parsing: each line is split by tab and it is extracted to nickname, fullname, email, fcc, and comment; input validation on each line is done.
- Whether the contact is a distribution list or a single contact is determined; if the distribution list contains nick names, they are replaced with email addresses.
- If the extracted info has non-empty nickname or fullname, the contact is exported otherwise the data is ignored.
- The extracted fields are written to the output in the ldif format described above.

**Security measures:**

- The script is run under tainted mode and therefore Perl takes special precautions called taint checks to prevent both obvious and subtle traps. Detailed information about tainted mode is given in [7].
- Htdocs folder of the webserver has no executable permissions, so attacks are discouraged.

- Writing to temporary file is avoided by using the perl IO:String class to channel the output to a string instead of a file. When using a temporary file, a local attacker could create symbolic links in the temporary file directory, pointing to a valid file somewhere on the file system. When a Perl script is executed, this would result in the file being overwritten with the rights of the user running the utility, which could be the root user [8].

**Command line script:** A command line script is also available. It uses two arguments: the pine address book and the output ldif file. When run without arguments it displays the usage. When run with the "-h" option, it displays more detailed help information for the user.

## 5 Conclusions

Our research proves that a good anti-spam solution can be offered by a combination of conservatively configured server level spam filters, combined with a client featuring learning filters. The best clients for our heterogeneous environment were the ones working on as many platforms as possible, i.e. mozilla thunderbird or seamonkey. The most difficult task of the presented scenario is to migrate users from old mail clients (e.g. pine) to thunderbird. The paper presents a home-grown solution to migrate addressbooks, that fit most of our users needs.

### Acknowledgments

Authors would like to thank Computing Services at Carnegie Mellon University for providing useful data and statistics that made this work possible.

## References

[1] CMU Computing Services, *How the Andrew Spam Filter Works*, http://www.cmu.edu/computing/doc/email/spam/how.html

[2] InterGuru, *Interguru's Pine to Ldif Address Conversion*, http://www.interguru.com/pineldif.htm

[3] Johannes Becker, *Addressbook Converter*, http://www.uni-giessen.de/hrz/kommuni/ldap/a_book.shtml

[4] The Fink Project, *Package abook-0.5.6-1002*, http://pdb.finkproject.org/pdb/package.php/abook

[5] Joe Casad, "Studies in Software as a Service", *Linux magazine*, Vol. 89, pp. 21-23, 2008.

[6] Network Working Group, *The LDAP Data Interchange Format (LDIF) - Technical Specification*, http://tools.ietf.org/html/rfc2849

[7] CPAN, *perlsec - Perl security*, http://search.cpan.org/dist/perl/pod/perlsec.pod

[8] Gentoo Linux, *Perl: Insecure temporary file creation*, http://www.gentoo.org/security/en/glsa/glsa-200412-04.xml