A Project Report on

Mail Content Filtering

Submitted to:

Visveswaraiah Technological University

Belgaum

By:
Debdas Ghosh (1SI99CS018)
Gesly Abraham George (1SI99CS024)
Ramya P.M (1SI99CS061)
Sagnik Sahu (1SI99CS065)

In partial fulfillment of the requirements for the award of Bachelor of Engineering in Computer Science and Engineering

Project Work carried out at

National Aerospace Laboratories
Bangalore - 560017

Under the guidance of:

Mr. N. Kumar
Scientist 'E2'
Deputy Head, Computer Support and Services
National Aerospace Laboratories
Bangalore - 560017

Siddaganga Institute of Technology

Department of Computer Science & Engineering

Tumkur-572103, Karnataka

June 2003

DECLARATION

We hereby declare that the entire work embodied in this dissertation has been carried out by us and no part of it has been submitted for any degree or diploma of any institution previously.

Signatures of the students

Date

Place

Debdas Ghosh

Gesly Abraham George

Ramya P.M.

Sagnik Sahu

Certificate

This is to certify that the project report entitled "**Mail Content filtering**" by

Debdas Ghosh
Gesly Abraham George
Ramya P.M.
Sagnik Sahu

in partial fulfillment of the award of the degree of Bachelor of Engineering in Computer Science & Engineering **Visveswaraiah Technological University**, **Belgaum** is a bonafide record of work carried out at National Aerospace Laboratories, Bangalore from 24th February to 30th June 2003 under my supervision and guidance.

Signature of the Guide

Mr. N. Kumar
Scientist 'E2'
Deputy Head,
Computer Support and Services

Date:

SIDDAGANGA INSTITUTE OF TECHNOLOGY

TUMKUR - 572 103

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Certificate

Certified that the project work entitled *Mail Content Filtering* is a bonafide work carried out by

1. **Mr. Debdas Ghosh**
2. **Mr. Gesly Abraham George**
3. **Ms. Ramya P.M.**
4. **Mr. Sagnik Sahu**

in partial fulfillment for the award of degree of Bachelor of Engineering in Computer Science & Engineering of the *Visveswaraiah Technological University*, Belgaum during the year 2003. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report deposited in the departmental library. The report has been approved as it satisfies the academic requirements with respect to project work prescribed for the Bachelor of Engineering degree.

Mr. B. Satish Babu, M.E.	**Dr. B.B. Amberker,** Ph.D., MIEEE	**Dr. M.N. Channabasappa,** MA, Ph.D.
Associate Professor	Professor & Head	Principal

Debdas Ghosh (1SI99CS018)
Gesly Abraham George (1SI99CS024)
Ramya P.M. (1SI99CS061)
Sagnik Sahu (1SI99CS065)

Name of the examiners	**Signature with date**
1.	
2.	

ACKNOWLEDGEMENT

We express our sincere thanks to **Dr. B.R. Pai**, **Director**, **National Aerospace Laboratories** (**NAL**), **Bangalore** for permitting us to carry out this project work at NAL.

We are grateful to **Dr. S. Ramamurthy**, **Head**, **Technical Secretariat**, **NAL** for his co-operation during the term of the project.

We wish to express our gratitude to **Dr. C. Dinah Geethalathai**, **Head**, **Computer Support and Services** (**CSS**), **NAL** for permitting us to carry out the project at the CSS division.

Most importantly, we are beholden to **Mr. N. Kumar**, **Deputy Head**, **Computer Support and Services**, **NAL**, our guide for the project. But for his motivation and encouragement, it would have been extremely difficult to work with diligence and keep up with the deadlines. We are deeply indebted for the trust he placed in us and we hope to have lived up to his expectations.

We are deeply grateful to **Mr. S. Nagaraj** and **Mr. P. Kripakaran**, **Scientists of CSS division** for their help in setting up the environment for the working and testing of the project.

We would also like to thank **Mr. Joseph Udaya Kumar**, **Mr. G. Gururaja** and **Mr. Nagendra Kumar**, **Scientists of CSS division**, who were always ready to lend a helping hand by providing the necessary infrastructure to complete this project.

We are thankful to **Dr. M. N. Channabasappa**, **Principal**, **Siddaganga Institute of Technology**, **Tumkur** for giving us timely support and encouragement.

We express our profound gratitude to **Dr. B. B. Amberker**, **Professor and Head**, **Department of Computer Science and Engineering**, **Siddaganga Institute of Technology**, **Tumkur** who has always been a source of inspiration.

We wish to convey our heartfelt gratitude to **Mr. B. Sathish Babu**, **Associate Professor**, **Department of Computer Science and Engineering**, **Siddaganga Institute of Technology**, **Tumkur** for his valuable support and insightful tips.

The project would not have taken shape but for the timely suggestions and clarifications by **Dr. Neil W. Rickert**, **Professor of Computer Science**, **Northern Illinois University**. Dr. Rickert is a leading Sendmail guru and the co-author of *Sendmail 1st Edition, O'Reilly Publications.*

In the same strain, we express our gratitude to **Mr. Pramod Perumparambil**, **Vice Chairman**, **Iormyx Software Solutions**, **Bangalore** who helped us throughout the project, directing us towards the final goal.

We also express our gratitude to **Mr. Vidyashankar**, **Infosys Technologies**, **Bangalore** for all the help and pointers given.

Project Associates

Debdas Ghosh

Gesly Abraham George

Ramya P.M.

Sagnik Sahu

SYNOPSIS

.

Generally leakage of sensitive information from any organization may lead to crisis. It may lead to a national crisis especially in case of government organization. **National Aerospace Laboratories (NAL)** being an institution where projects of national importance are undertaken it becomes mandatory to adopt measures to prevent the leakage of sensitive information. It is of prime importance to monitor all the information that may go out of the institution, at every level of the organizational hierarchy.

In this world empowered by the advent of information technology, one of the most popular and convenient means of communication is through the use of e-mails. The leakage of information may take place in the form of mails sent out of the NAL mail server. This project intends to monitor the mails that go out of the NAL mail system. Each mail that goes through the mail server is captured, parsed for sensitive information and action is taken as per the result.

Confidential information may be present in the body of the mail or the attachments that are sent along with a mail. It is generally accepted that this breach can be prevented if the body of the mail is parsed for sensitive words. The attachments may have different formats - .docs, .txt, .pdf, .exe, .gif, .jpeg, .vbs, etc. Some of these formats are considered malicious and harmful. Mails found to have such attachments are immediately blocked.

For any given organization, considering the projects handled a few words would be considered as sensitive. The system administrator maintains a file of such keywords. The body of the mail and the attachment is searched for these keywords. If the mail is beyond suspicion, it is sent through. If the body or the attachment of the mail contains any sensitive keyword that the employee is not allowed to use, appropriate action is taken. The mail is blocked and a mailer-daemon message is sent to the employee informing the person that the mail could not be sent because it could result in leakage of sensitive information. The administrator is also sent a copy of the offensive mail. This not only ensures no information is leaked, but also that the authorities are aware of the transgression. An employee found to be guilty of breach of confidence is liable for legal action.

Table of Contents

1. Introduction ……………………………………………………………………………….1
- 1.1 Project Overview
  - 1.1.1 Scope of the Project
  - 1.1.2 Present Development
- 1.2 Organizational Profile
  - 1.2.1 About National Aerospace Laboratories
  - 1.2.2 About Computer Support and Services Division

2. Literature survey………………………………………………………………………..6
- 2.1 The E-mail System
  - 2.1.1 Overview of Mail Delivery
- 2.2 Mail User Agent (MUA)
- 2.3 Mail Transfer Agent (MTA)
- 2.4 Mail Delivery Agent (MDA)
  - 2.4.1 The Mail Hub and Delivery Agents
  - 2.4.2 Define a Mail Delivery Agent
- 2.5 Sendmail
  - 2.5.1 Three Important Parts
  - 2.5.2 Run Sendmail by hand
  - 2.5.3 Roles played by Sendmail
  - 2.5.4 How to run Sendmail
  - 2.5.5 The Sendmail.cf File
  - 2.5.6 The Sendmail.mc File
  - 2.5.7 The Sendmail MTA
  - 2.5.8 Filtering Mail with Milter

3. System Analysis and Requirements ……………………………….………………26
- 3.1 Problem definition
- 3.2 Existing system
- 3.3 Proposed System
- 3.4 Software Requirement Specification
  - 3.4.1 Introduction
  - 3.4.2 General Description
  - 3.4.3 Specific Requirements
- 3.5 Hardware requirements
- 3.6 Software requirements

4. System Design…………………………………………………………………………34
- 4.1 Feasible Solutions
  - 4.1.1 Processing the mail in the queue
  - 4.1.2 Altering the Sendmail source code
  - 4.1.3 Altering the sendmail configuration file
  - 4.1.4 Altering the sendmail.mc file
  - 4.1.5 Using Mail Filter APIs
  - 4.2 Adopted Solution
  - 4.2.1 Advantages of using Milter
- 4.3 Overall System Description

4.3.1 Overview of the present set up of the mail-server
4.3.2 Different components used in the present system
4.3.3 The component where the mail filtering software resides
4.3.4 Brief description of the working of the mail-filter
4.3.5 Design of the database maintained
4.4 Control-Flow Diagram
4.5 User Interface Design
4.5.1 Sfilter Configuration Tool
4.7 Development standard
4.6 Design constraints

5. Detail Design……………………………………………………………………...45
5.1 Design Description
5.1.1 Grep
5.1.2 Sed
5.1.3 Awk
5.2 Module Decomposition
5.2.1 Processing the headers
5.2.2 Processing the body of the mail
5.2.3 The Filter module

6. Development tools…………………………………………………………………...49
6.1 Ripmime
6.2 Antiword
6.3 Glade
6.4 Unzip
6.5 Tar
6.6 Gunzip

7. Implementation……………………………………………………………………...51
7.1 Configuring filter to work in sync with Sendmail
7.1.2 Communication between sendmail and the mail-filter
7.1.3 Running the mail-filter as a daemon process during system boot
7.1.4 Warning

8. Testing…………………………………………………………………………………53
8.1 Unit testing

9. Conclusion……………………………………………………………………...55

10. Future Enhancements………………………………………………………..56

References……………………………………………………………………………...57

Appendix………………………………………………………………………………58
A: Sample Screens
B: Installation Guide
C: Definitions, Acronyms, Abbreviations
D: Control File Templates

1. INTRODUCTION

1.1 Project Overview

The hierarchy that exists in our society is primarily dependent on the comparison of the wealth accrued by the individuals. This wealth has taken different forms over the decades. There was an age when the most valuable assets were in the form of material wealth. In order to protect these assets, physical barriers are required. There exist laws that help a person lay claim on such wealth. Land is demarcated with fencing; money and gold, which has always been the measure of wealth, is stacked away in inaccessible places.

On a higher level, national security largely involved maintaining tight vigil along the border of the country. Anything and anyone crossing the border are thoroughly scrutinized. The steps taken by nations to ensure this security have been quite controversial and this has lead to a number of ethical conflicts.

Somewhere down the line, the form taken by assets began to change. Slowly, material wealth started to take a back seat to intellectual property. In order to guard this new kind of wealth, the conventional methods of physical protection became obsolete. New methodologies had to be devised to achieve the required security. Since the human mind tends to improvise by extending already existing systems to fit his needs, he devised logical barriers for this intellectual property. These protective measures take a more complicated form, considering the vast scope and obscurity involved.

National security has also widened its scope. The information that goes out of any institution of the country has become significant. Every country has a number of organizations whose intellectual assets are far superior to the physical resources. It is an accepted fact that it just takes an intelligent human mind to recreate a contraption of any kind, if it has access to the basic working principle.
For many research organizations like NAL, preventing the leakage of confidential information is the key to their success. E-mail is an easy outlet for accidental or deliberate leaks of confidential information.

The proposed system entitled – **Mail Content Filtering** expects to perform content filtering on the mails going out through the NAL mail server. Content Filtering can be achieved by monitoring the content of the messages leaving the organization. A successful implementation of the filter will help to avoid leakage of confidential information from the organization through the mail system of the organization.

1.1.1 Scope of the Project

Securing sensitive information is a key concern for any organization. Sensitive information may take various forms depending upon the organization in question and the work that is carried out in that particular organization. Without doubt, any communication that takes place by an employee of an organization with the outside world may result in divulging information that is considered to be confidential to the organization. This leakage of information may be quite unintentional on the part of the person communicating. Unfortunately, organizations are also threatened by a deliberate leakage of information by some miscreants. In either case, there should not only be a system existing that prevents this leakage, but also informs the administrator of the breach of confidence that has taken place.

NAL is a national organization that deals with the development of aerospace technologies. Its main area of work is to design and develop small and medium-sized aircraft for the civil sector.

It is quite obvious that the kind of information that may leak out of the organization might not only affect the reputation and the integrity of the organization, but may also affect national security. This being a very momentous issue, lot of care has to be taken to prevent all such misdemeanors.

In the present world, emails are the most convenient form of communication. Email is probably the most popular Internet related service. Email provides us with speedy communication facilities. It also allows us to easily send files, from simple text documents to spreadsheets; images to video clips. It is outside the scope of this document to list the advantages of the email system over conventional methods of communication. It can be mentioned, however, that emails have slowly become the most commonly used means of communication in any organization, be it commercial or research-oriented.

The mail system is highly autonomous and can provide a lot of confidentiality to the user. This is very advantageous in many ways. However, this could also mean that it is highly difficult for an organization to monitor the activities of its employees and prevent any untoward communication. This has resulted in a lot of apprehension on the security of the system. The thin line that exists between the rightful claims of an organization on its employee’s communication with the outside world to the intrusion of the employee’s personal space is a highly controversial issue.

As a middle ground, some organizations insist that an employee can only use the mail services that the official website provides. NAL also provides its employees with an account in its mail server. In order to prevent the intentional or unintentional leakage of information, a system has to be designed to monitor the mails that are sent out of the organization using these accounts. All the communication that takes place over this system has to be monitored and any indication of leakage of sensitive information has to be immediately blocked and the higher officials should be notified.

The system that is designed does precisely this. A list of keywords that translate as sensitive information is stored and each and every mail that goes out of the NAL mail server is parsed for these words. If the mail or the attachment that is sent with the mail contains any sensitive keyword, the mail is blocked and a copy of the same is sent to the administrator.

1.1.2 Present Development

A Mail User Agent (MUA) is any program that users run to read, reply to, compose and dispose of email. A Mail Transfer Agent (MTA) is a highly specialized program that delivers mail and transports it between machines. “Sendmail” is the most popular MTA in the Linux platform. Almost every client program (Mail User Agents) sends emails using Sendmail. Sendmail is the MTA used to transfer the mail going through the NAL mail server. Our objective is to insert our filter into the appropriate places in the “Sendmail” MTA.

The filter designed by us scrutinizes the mail that is handed over to the MTA. This filter then parses the body of the mail and the attachment that is sent along with the mail. The filter has to check for sensitive information.

The mail contains not only the body of the mail. It is also possible that an attachment may be sent along with the mail. This attachment may take different forms. It may be a text document, a Microsoft Application, an image, a sound file or a video clip, a binary file, a compressed file, etc. We maintain a list of types of files that may be further processed. If the attachment is of a type that cannot be identified, the mail is blocked. If the attachment is of text format, the body of the attachment is also considered for parsing. If the file is compressed, it is decompressed and searched for sensitive information.

The filter scans the body and the attachment of the email for some pre-defined keywords. All the keywords that are considered sensitive are maintained in a file. Each user of the mail system belongs to one or more groups. Each group of the system has a file maintained that have a few keywords that the user of the group is permitted to use.

From these two lists, a set of keywords that are considered to be sensitive for a particular user is formulated. The mail is scanned for these words. If the person is found to have used a particular sensitive keyword, the mail is blocked. The employee is sent a mail informing him that the mail could not be delivered to the intended recipient because he is not authorized to send that particular mail. Besides this, the administrator is sent a copy of the offending mail. This will ensure that the appropriate action may be taken to prevent the further occurrence of such an event.

NAL system usually uses Pine as the default MUA. Sendmail is the default MTA used by NAL mail server. Whenever a new user is created in the NAL mail server, he is assigned an email-id that takes the form *user@nal.res.in* and *user@css.cmacs.ernet.in*. The user is hence provided with two mail-ids. The official NAL website *http://www.nal.res.in* also provides the user with a user-friendly interface to send the mail and to access his mailbox.

Till this package was developed, having full trust on its employees a Mail Content Filtering facility did not exist in the NAL mail server. However the existing outside scenario warrants the development and implementation of a filter to prevent untoward happenings. The employee who has an account in the system is free to send any kind of information and any kind of attachment file to an address of his choice. This led to some concern as to the safety of such a system, resulting in the need for a foolproof system that will prevent such mishap and leave no scope for defraud.

1.2 Organizational Profile

1.2.1 About National Aerospace Laboratories

National Aerospace Laboratories (NAL), a constituent of Council of Scientific and Industrial Research (CSIR), is India's pre-eminent civil R&D establishment in aeronautics and allied disciplines. NAL was set up at Delhi in 1959 and moved to Bangalore in 1960.

NAL's primary objective, as articulated in its new Vision Statement, is the "development of aerospace technologies with a strong science content and with a view to their practical application to the design and construction of flight vehicles". NAL is also required "to use its aerospace technology base for general industrial applications".

NAL's core competence spans practically the whole aerospace spectrum. Over the years, NAL has made very significant contributions to all Indian aerospace programs; often even setting the national agenda for such programs. During the last decade NAL has spearheaded the effort to design and develop small and medium-sized aircraft for the civil sector.

NAL's real strength lies in its vast reservoir of expertise and facilities created over the years. With this imposing infrastructure, NAL has been very successful in obtaining a large number of R&D contracts for testing and subsystem development for various national programs as well as industries all over India and abroad. In the past decade (1987-97), NAL undertook approximately 400 projects worth about 60 million US$. Over the last few years, NAL has earned more than 60% of its budget through external resources, a unique achievement for CSIR laboratories.

NAL is well equipped with modern and sophisticated facilities, which include national facilities like the Nilakantan Wind Tunnel Center and the computerized full-scale fatigue test facility. The various facilities and multi-disciplinary expertise, developed primarily for the aerospace sector, are also utilized in other sectors involving high technology. NAL is recognized as a center for failure analysis and extends its support in investigating failures and accidents both for aerospace and other general facilities. Other major facilities at NAL include: the acoustic test facility, turbo machinery and combustion research facilities, Composite Structures Laboratory, black box readout systems and the FRP fabrication facility. NAL has staff strength of about 1300 with about 350 full-fledged R&D professionals (over 100 Ph.D.'s). It is thus in a unique position to offer R&D support, expertise and services to both aerospace and non-aerospace sectors of industry. Some major recent contracts include: development of carbon fiber composite wings for India's Light Combat Aircraft (LCA) program, design, development and fabrication of a fully-automated autoclave for Hindustan Aeronautics Limited (HAL), development of co-cured fin and rudder for LCA and a shake test facility for HAL's Advanced Light Helicopter (ALH).

Spin-off technologies from aerospace R&D activities have significantly contributed to the non-aerospace sector everywhere in the world. Conscious of this aspect, NAL has made special efforts to identify those developments, which could result as offshoots from the main R&D programs. About 30 such technologies developed over the last decade have been successfully licensed and transferred to 54 industries against a premier value of 100,000 US$. The cumulative production value of these technologies is over 10 million US$.

NAL's models for business development activities include in-house projects leading to commercialization, sponsored projects, industry-lab linkages, multi-agency collaborative projects and international contracts. During the last 24 months, NAL has obtained 12 contracts worth over 25 million US$. NAL has also undertaken about a dozen international projects for Boeing, USA; Civil Aviation Authority, UK; IBM Corporation, USA; Hitachi, Japan etc.

NAL has therefore come a long way from its modest beginnings in 1959-60 when it was housed for some time in the stables of a former Maharaja's palace in Bangalore. This development has been possible because of the vision and commitment of its former Directors: Dr P Nilakantan, Dr S R Valluri, Prof R Narasimha, Dr K N Raju and Dr T S Prahlad.

1.2.2 About Computer Support and Services Division

The Computer Support and Services Division (CSS) caters to the needs of the various divisions of NAL and outside organizations by providing technical/infrastructural support in the areas of Uninterrupted Power Supply, AC, EPABX-DID facility, Computing facilities, Networking, Internet and E-mail facilities, PC maintenance and Training programs.

In addition, the two major goals of CSS are (a) Establishment of a campus-wide Fiber Distributed Data Interface (FDDI) Network with internet connectivity and (b) Grooming the

Advanced Information Technology Center (AITC), set up under the CTD/USAID program, into an excellent software development and training center.

1.2.2.1 Campus Network Program:

A Linux based e-mail server has been set up at CSS. This is being used extensively by more than 180 users through the campus network. This e-mail server can also be accessed through the campus network. This e-mail server can also be accessed through the dial-up connection from outside NAL and also by the staff who are not on the campus network.

In order to provide connectivity among all the CSIR labs, a satellite-based closed user-group computer communication network called CSIRNET is being established with VSATs at each of the CSIR labs, a central hub t CSIR headquarters with gateway to Internet and leased transponder space on any of the satellites. The VSATs are of the SCPC-DAMA type of 64 kbps speed. This would provide full mesh connectivity of voice, fax and data among the laboratories and at the same time a high speed access to the Internet through a 2 X 64 kbps leased line connection from Delhi, headquarters to GIAS of VSNL.

The VSAT has been connected to the campus network and is working satisfactorily for Internet Access.

1.2.2.2 Advanced Information Technology Center:

The Advanced Information Center (AITC) set up at CSS under the CTD/USAID program, comprises of the Geographic Information System (GIS) Center, CAD and AutoCAD Centers. The objectives being (a) to provide excellent training in the areas of GIS, CAD, virtual reality, animation, database management applications, Mathematical Modeling and advanced computer simulation to personnel from educational institutions, R&D organizations, government offices, CSIR labs and women entrepreneurs and (b) to make AITC self supporting by making AITC an excellent software development center.

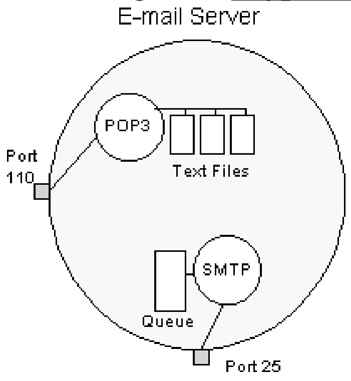
2. Literature Survey

A Mail User Agent (MUA) is any program that users run to read, reply to, compose and dispose of email. A Mail Transfer Agent (MTA) is a highly specialized program that delivers mail and transports it between machines. “Sendmail” is the most popular MTA in the Linux platform. Sendmail is used to transfer the mail going through the NAL mail server. Our objective is to insert our filter into the appropriate places in the “sendmail” MTA. The filter will be written in the C language. The filter has the following functionalities.

The filter has to check the content of the mail for sensitive information. This is achieved as follows. We maintain a file of keywords that are sensitive for the institution. The employees of the organization are categorized into groups. A file is maintained for each of these groups. This contains a list of all those sensitive keywords that a member of the group has authority to access. From this, we deduce a list of words that a user is not allowed to use. The mail is then scanned for these words. If any such word is found, the mail is blocked and the user is notified.

2.1 The E-mail System

The *E-MAIL SYSTEM* consists of two different servers running on a server machine. One is called the *SMTP Server*, where SMTP stands for Simple Mail Transfer Protocol. The SMTP server handles *outgoing mail*. The other is a *POP3 Server*, where POP stands for Post Office Protocol. The POP3 server handles *incoming mail*. ***<u>A typical e-mail server looks like this</u>***



The SMTP server listens on port 25 while the POP3 server listens on port 110.

2.1.1 Overview of Mail Delivery

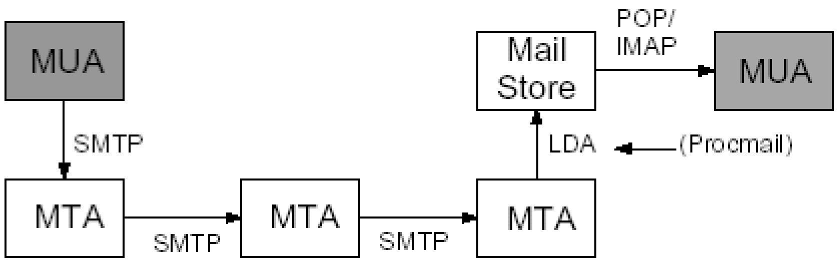


Figure: The various parts of mail delivery system

The various parts of mail delivery are

- **MUA** - Mail User Agent. Program that users run to read, reply to, compose and dispose of email.
  Examples: pine, elm, mutt

- **MTA** - Mail Transfer Agent. Program responsible for receiving, routing, and delivering e-mail messages. MTAs receive e-mail messages and recipient addresses from local users and remote hosts, perform alias creation and forwarding functions, and deliver the messages to their destinations. An MTA is sometimes called a mail transport agent, a mail router, an Internet mailer, or a mail server program. Commonly used MTAs include sendmail, qmail, and Exim.

- **Mail Store** - Where mail is stored until it is read. Usually in /var/spool/mail

- **LDA** - Local Delivery Agent. Program that performs the final delivery into the mail store

- **POP** - Post Office Protocol. A method for retrieving mail from a mail store
- **IMAP** - Internet Message Access Protocol. A better protocol than POP/POP3.

- **SMTP** - Simple Mail Transfer Protocol. The language used by MTA's to talk to each other.

2.2 Mail user agent (MUA)

A mail user agent (MUA) is any of the many programs that users run to read, reply to, compose, and dispose of email. Examples of an MUA include the original UNIX mail program (*/bin/mail*); the Berkeley *Mail* program; its System V equivalent (*mailx*); free software programs such as *mush*, *elm*, and *mh*; and commercial programs such as *Zmail*. Many MUAs may exist on a single machine. MUAs sometimes perform limited mail transport, but this is usually a very complex task for which they are not suited.

2.3 Mail Transfer Agent (MTA)

A mail transfer agent (MTA) is a highly specialized program that delivers mail and transports it between machines, like the post office. Usually, there is only one MTA on a machine. The *sendmail* program is an MTA. Others include *MMDF*, *Smail 3.x*, and *Zmailer*.

The MTA itself is split into four "engines".

- The *Listener* accepts responsibility for delivering incoming messages from any SMTP-compliant mail agent. It also provides SMTP authentication, Transport Layer Security (TLS) encryption, and a content management API that provides a link from within the MTA to third party content filtering software.

- The *Parsing and Routing Rules* engine takes care of any tricky addressing issues - resolving aliases, modifying addresses, detecting errors and spam mail – before moving the message into the queue for delivery.

- Sendmail can handle multiple *Message Queues* that hold messages until the receiving agent is ready to accept delivery.

- The *Delivery engine* then finds the receiving mail agent, rewrites the e-mail message in the format of the delivery agent and sends it.

2.4 Mail Delivery Agent (MDA)

2.4.1 The Mail Hub and Delivery Agents

Instead of having each individual workstation in a network handle its own mail, it can be advantageous to have a powerful central machine that handles all mail. Such a machine is called a *mail hub* our workstations can be thought of as the outlying offices (client machines) and your central mail-handling machine as the hub. The machines perform the following functions for the entire site:

All incoming mail is sent to the hub, meaning that no mail is sent directly to a client machine. This hub approach has several advantages. No client needs to run a *sendmail* daemon to listen for mail. No client's name needs to be known to the outside world, thus insulating client machines for easier security.

Rather than having each client send its mail directly to the outside world, all outgoing mail from clients is sent to the hub, and the hub then forwards that mail to its ultimate destination. The advantages are that clients do not need to be continually aware of changes in the Internet; deferred mail queues on the hub, not on the client machine, making management simpler; and a single, simple *sendmail.cf* file may be shared by, or distributed to, all the clients.

All outgoing mail is modified so that it appears to have originated on the hub. The alternative is to have *reply* mail returned directly to each client. The advantages of the hub approach are that all mail appears to come from a single, giant machine (making replying to mail easier).

All mail to local users is delivered to, and spooled on, the hub. The alternative is to have one or more separate mail spool directories on separate client machines. The advantages of the hub approach are that all local delivery is handled by one machine, that no client needs to accept mail for local delivery, and that management is easier with a single spool.

There are disadvantages to the hub approach when a site is composed of differing machine architectures and when machines are spread over many networks:

At sites where there is a huge amount of mail constantly flowing, the load on the hub can become excessive. This can cause client mail to be queued rather than sent, even when mail is destined for local delivery.

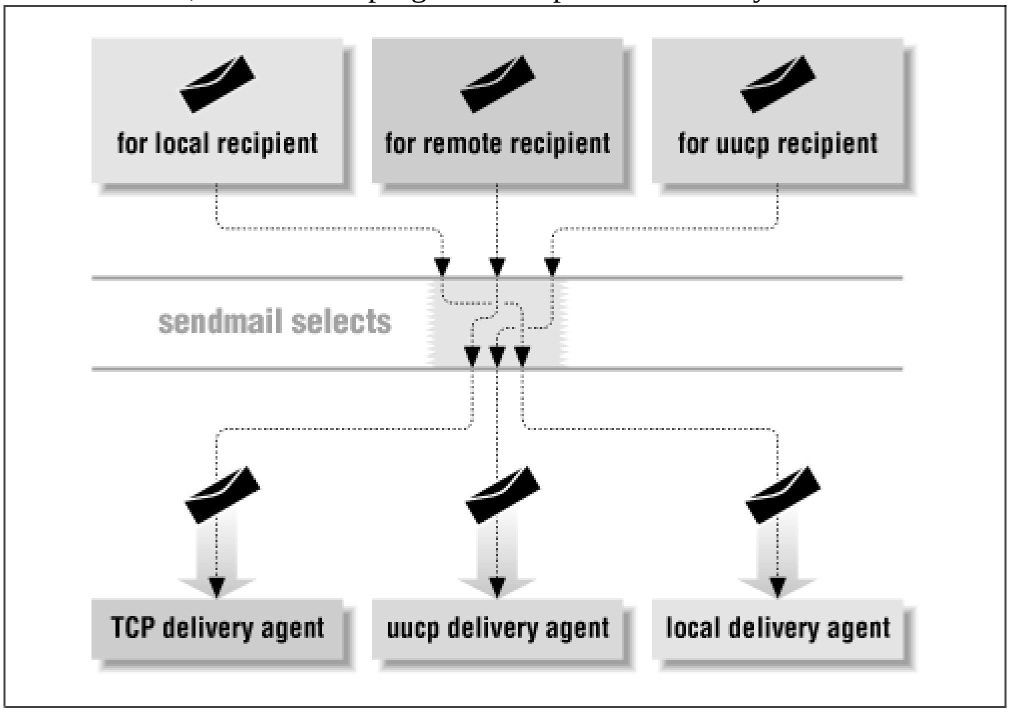
For the hub to work, it needs to know the login names of all users on the system. Either it must have a master */etc/passwd* file that is the union of all systems' *passwd* files, or it must use Network Information Services (NIS) to access such a master file. [2] In the absence of universal user *passwd* knowledge, it must have an *aliases* file entry for every local user.

Because all mail passes through the hub, rather than being sent to the recipient directly, there are unavoidable delays. Those delays are negligible at small sites (a couple of hundred or so users) but can become significant at sites with a huge number of users.

If a client machine has direct UUCP connections or is connected to multiple networks, a more complex configuration file is needed.

2.4.2 Define a Mail Delivery Agent

Other than forwarding mail messages over a TCP/IP network, *sendmail* does not handle mail delivery itself. Instead, it runs other programs that perform delivery.



2.5 Sendmail

“Sendmail” is the most popular MTA in the Linux platform. In its simplest role, that of transporting mail from a user on one machine to another user on the same machine, *sendmail* is almost trivial. All vendors supply a *sendmail* (and configuration file) that will accomplish this.
On hosts that are connected to the Internet, *sendmail* should use the Domain Name System (DNS) to translate hostnames into network addresses. Machines with UUCP connections, on the other hand, need to have *sendmail* run the *uux* program.
The *sendmail* program needs to transport mail between a wide variety of machines. Consequently, its configuration file is designed to be very flexible. This concept allows a single binary to be distributed to many machines, where the configuration file can be customized to suit particular needs. This configurability contributes to making *sendmail* complex.

Consider, for example, when mail needs to be delivered to a particular user. The *sendmail* program decides on the appropriate delivery method based on its configuration file. One such decision process might include the following steps:

- If the recipient receives mail on the same machine as the sender, deliver the mail using the */bin/mail* program.
- If the recipient's machine is connected to the sending machine using UUCP, use *uux* to send the mail message.

- If the recipient's machine is on the Internet, the sending machine transports the mail over the Internet.
- Otherwise, the mail message may need to be transported over another network (such as BITNET) or possibly reject.

2.5.1 Three Important Parts

The *sendmail* program is actually composed of several parts, including programs, files, directories, and the services it provides. Its foundation is a configuration file that defines the location and behavior of these other parts and contains rules for rewriting addresses. A *queue directory* holds mail until it can be delivered. An *aliases file* allows alternative names for users and creation of mailing lists.

2.5.1.1 The Configuration File

The configuration file contains all the information *sendmail* needs to do its job. Within it you provide information, such as file locations, permissions, and modes of operation.

Rewriting rules and rule sets also appear in the configuration file. They transform a mail address into another form that may be required for delivery. They are perhaps the single most confusing aspect of the configuration file. Because the configuration file is designed to be fast for *sendmail* to read and parse, rules can look cryptic:

```
R$+@$+          $:$1<@$2>    focus on domain
R$+<$+@$+>      $1$2<@$3>    move gaze right
```

2.5.1.2 The Queue

Not all mail messages can be delivered immediately. When delivery is delayed, *sendmail* must be able to save it for later transmission. The *sendmail* queue is a directory that holds mail until it can be delivered. A mail message may be queued:

- When the destination machine is unreachable or down. The mail message will be delivered when the destination machine returns to service.

- When a mail message has many recipients. Some mail messages may be successfully delivered, and others may not. Those that fail are queued for later delivery.

- When a mail message is expensive. Expensive mail (such as mail sent over a long-distance phone line) can be queued for delivery when rates are lower.

- When safety is of concern. The *sendmail* program can be configured to queue all mail messages, thus minimizing the risk of loss should the machine crash.

2.5.1.3 Aliases and Mailing Lists

Aliases allow mail that is sent to one address to be redirected to another address. They also allow mail to be appended to files or piped through programs, and they form the basis of mailing lists. The heart of aliasing is the *aliases* file (often stored in database format for swifter lookups). Aliasing is also available to the individual user via a file called *.forward* in the user's home directory.

2.5.2 Run Sendmail manually

Most users do not run *sendmail* directly. Instead, they use one of many mail user agents (MUAs) to compose a mail message. Those programs invisibly pass the mail message to *sendmail*, creating the appearance of instantaneous transmission. The *sendmail* program then takes care of delivery in its own seemingly mysterious fashion. Although most users don't run *sendmail* directly, it is perfectly legal to do so. Many system managers may need to do so to track down and solve mail problems.

Here's a demonstration of one way to run *sendmail* manually. First create a file named *sendstuff* with the following contents:

This is a one line message.

Second, mail this file with the following command line, where *you* is your login name:
% /usr/lib/sendmail you < sendstuff

When you run sendmail, any command-line arguments that do not begin with a **-** character are considered to be the names of the people to whom you are sending the mail message. The mail message that you just sent consists of a header and body.

```
From  you@Here.US.EDU  Fri Dec 13 08:11:44 1996
Received:   (from you@localhost) by Here.US.EDU (8.8.4/8.8.4)
            id AA04599 for you; Fri, 13 Dec 96 08:11:44 -0700
Date:        Fri, 13 Dec 96 08:11:43
From:        you@Here.US.EDU (Your Full Name)
Message-Id: <9631121611.AA02124@Here.US.EDU>
To:          you

This is a one line message.
```

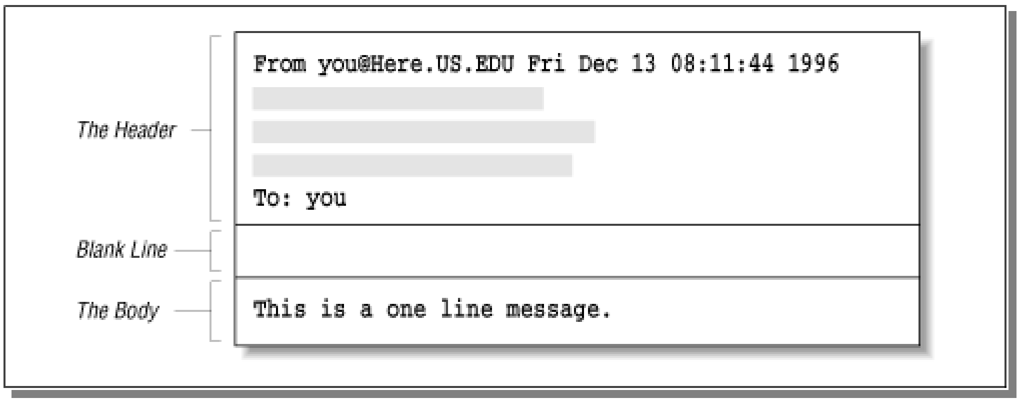

The Header
Blank Line
The Body
From you@Here.US.EDU Fri Dec 13 08:11:44 1996
To: you
This is a one line message.

Figure: Every mail message is composed of a header and a body

2.5.2.1 Header

Let's examine the header in more detail.

```
From  you@Here.US.EDU  Fri Dec 13 08:11:44 1996
Received:   (from you@localhost) by Here.US.EDU (8.8.4/8.8.4)
```

```
            id AA04599 for you; Fri, 13 Dec 96 08:11:44 -0700
Date:        Fri, 13 Dec 96 08:11:43
From:        you@Here.US.EDU (Your Full Name)
Message-Id: <9631121611.AA02124@Here.US.EDU>
To:          you
```

Most header lines start with a word followed by a colon. Each word tells what kind of information the rest of the line contains. There are many types of header lines that can appear in a mail message. Some are mandatory, some are optional, and some may appear many times. The line starting with the five characters "From " (the fifth character is a space) is added by some programs A Received: line is added each time a machine receives the mail message. (If there are too many such lines, the mail message will *bounce* and be returned to the sender as failed mail.) The indented line is a continuation of the line above, the Received: line. The Date: line gives the date and time when the message was originally sent. The From: line lists the email address and the full name of the sender. The Message-ID: line is like a serial number in that it is guaranteed to uniquely identify the mail message. And the To: line shows a list of one or more recipients. (Multiple recipients would be separated with commas.)

2.5.2.2 The Body

The body of a mail message consists of everything following the first blank line to the end of the file.

The Subject: header line is an optional one. The *sendmail* program passes it through as it is. Here, the Subject: line is followed by a blank line and then the message text, forming a header and a body. Note that a blank line must be truly blank. If you put space or tab characters in it, thus forming an "empty-looking" line, the header *will not* be separated from the body as intended.

2.5.2.3 The Envelope

To handle delivery to diverse recipients, the *sendmail* program uses the concept of an *envelope*. Information that describes the sender or recipient, but is not part of the message header, is considered envelope information. To illustrate what an envelope is, consider one way in which *sendmail* might run */bin/mail*, a program that performs local delivery:

```
 deliver to friend1's mailbox
       ↓
/bin/mail -d friend1        ← sendmail runs
           ↑
           the envelope recipient
```

Here *sendmail* runs */bin/mail* with a -d, which tells */bin/mail* to append the mail message to friend1's mailbox.
The envelope of a message for local delivery and the envelope of a message transported over the network may or may not contain the same information (a point we'll gloss over for now). In the case of */bin/mail*, the email message shows two recipients in its header:
To: friend1, friend2@remote ← *the header*
But the envelope information that is given to */bin/mail* showed only one (the one appropriate to local delivery):

-d friend1 ← *specifies the envelope*

Now consider the envelope of a message transported over the network. When sending network mail, *sendmail* must give the remote site a list of sender and recipients *separate from and before* it sends the mail message (header and body).

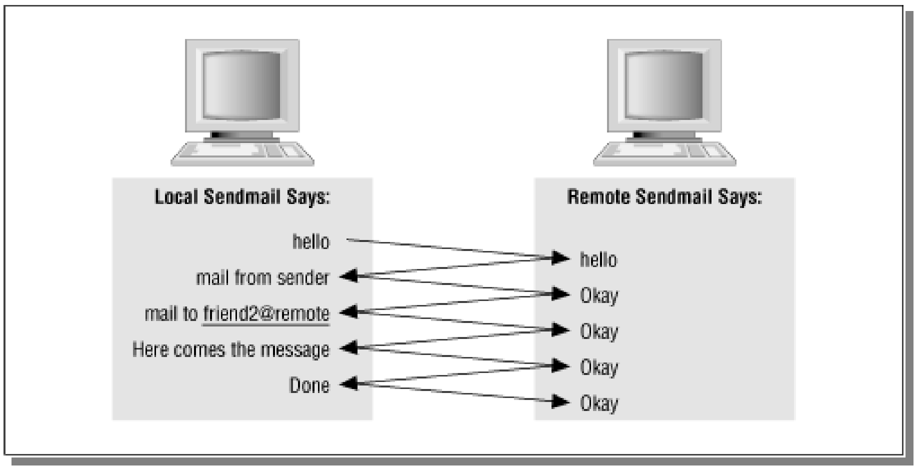


Figure: A simplified conversation

2.5.3. Roles played by Sendmail

The *sendmail* program plays a variety of roles, all critical to the proper flow of electronic mail. It listens to the network for incoming mail, transports mail messages to other machines, and hands local mail to a local program for local delivery. It can append mail to files and can pipe mail through other programs. It can queue mail for later delivery and understands the aliasing of one recipient name to another.

2.5.3.1 Role in the Filesystem

The *sendmail* program's role (position) in the local file system hierarchy can be viewed as an inverted tree.

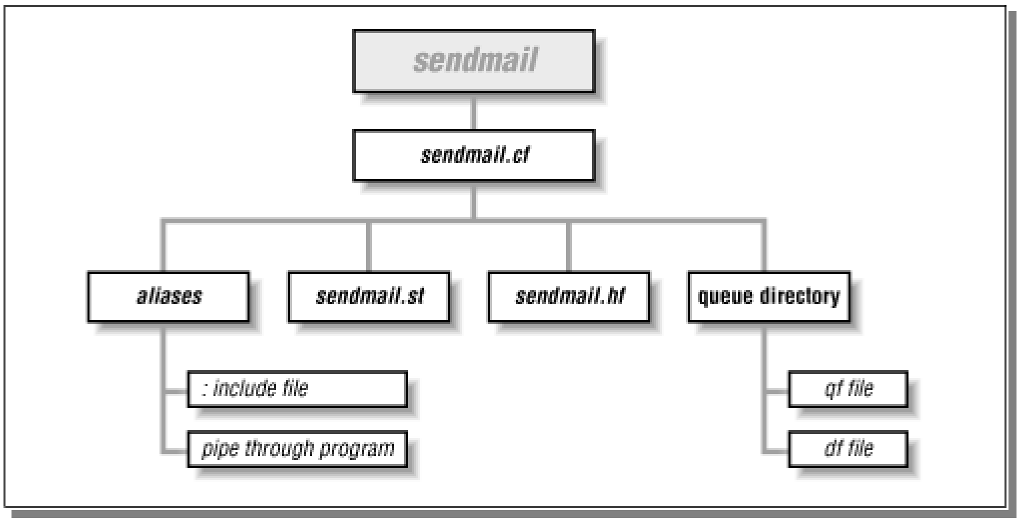


Figure: The sendmail.cf file leads to everything else

When *sendmail* is run, it first reads the */etc/sendmail.cf* configuration file. Among the many items contained in that file are the locations of all the other files and directories that *sendmail* needs. Files and directories listed in *sendmail.cf* are usually specified as full pathnames for security (such as */var/spool/mqueue* rather than *mqueue*).

2.5.3.2 The Aliases File

Aliasing is the process of converting one recipient name into another. One use is to convert a generic name (such as *root*) into a real username. Another is to convert one name into a list of many names (for mailing lists).

For every envelope that lists a local user as a recipient, *sendmail* looks up that recipient's name in the *aliases* file. (A local user is any address that would normally be delivered on the local machine. That is, *postmaster* is local, whereas *postmaster@remote* may not be.) When *sendmail* matches the recipient to one of the names on the left of the *aliases* file, it replaces that recipient name with the text to the right of the : character. After a name is substituted, the new name is then looked up, and the process is repeated until no more matches are found. When mail is bounced (returned because it could not be delivered), it is always sent from MAILER-DAEMON. That alias is needed because users may reply to bounced mail. Without it, replies to bounced mail would themselves bounce.

2.5.3.3 The Queue Directory

A mail message can be temporarily undeliverable for a wide variety of reasons, such as when a remote machine is down or has a temporary disk problem. To ensure that such messages are eventually delivered, *sendmail* stores them in its queue directory until they can be delivered successfully.

The QueueDirectory option in your configuration file tells *sendmail* where to find its queue directory:

```
O QueueDirectory=/var/spool/mqueue
```

When a mail message is queued, it is split into two parts, each part being saved in a separate file. The header information is saved in a file whose name begins with the characters qf. The body of the mail message is saved in a file whose name begins with the characters df.

2.5.3.4 Role in Local Delivery

Another role of the *sendmail* program is to deliver mail messages to local users. A local user has a mailbox on the local file system. Delivering local mail is done by appending a message to the user's mailbox, by feeding the mail message to a program, or by appending the message to a file other than the user's mailbox.

Delivery to a Mailbox

The configuration file line that begins with Mlocal defines how mail is appended to a user's mailbox file. That program is usually */bin/mail* but can easily be a program such as *deliver* or *mail.local*.

Under UNIX a user's mailbox is a single file that contains a series of mail messages. The usual UNIX convention (but not the only possibility) is that each message in a mailbox begins with the five characters "From&nbsp;" (the fifth is a blank space) and ends with a blank line.
The *sendmail* program neither knows nor cares what a user's mailbox looks like. All it cares about is the name of the program that it must run to add mail messages to that mailbox. In the example that program is */bin/mail*.

Delivery Through a Program

Mail addresses that begin with a | character are the names of programs to run.

ftphelp: "|/usr/local/bin/sendhelp"

Here, mail sent to the address ftphelp is transformed via an alias into the new address |/usr/local/bin/sendhelp. The | character at the start of this new address tells *sendmail* that this is a program to run rather than a file to append to. The intention here is that the program will receive the mail and do something useful with it.

The *sendmail* program doesn't run mail delivery programs directly. Instead, it runs a shell and tells that shell to run the program. The name of the shell is listed in the configuration file in a line that begins with Mprog

Mprog,   P=/bin/sh, F=lsDFMeu, S=10, R=20/40, D=$z:/,

2.5.3.5 Role in Network Transport

Another role of *sendmail* is that of transporting mail to other machines. A message is transported when *sendmail* determines that the recipient is not local. The following lines from a typical configuration file define delivery agents for transporting mail to other machines:

```
Msmtp,          P=[IPC], F=mDFMuX, S=11/31, R=21, E=\r\n, L=990,
Muucp,          P=/usr/bin/uux, F=DFMhuUd, S=12, R=22/42, M=10000000,
```

2.5.3.6 Role as a Daemon

Just as *sendmail* can transport mail messages over a TCP/IP-based network, it can also receive mail that is sent to it over the network. To do this, it must be run in *daemon* mode. A daemon is a program that runs in the background independent of terminal control.

As a daemon, *sendmail* is run once, usually when your machine is booted. Whenever an email message is sent to your machine, the sending machine talks to the *sendmail* daemon that is listening on your machine.

2.5.4 How to Run sendmail

2.5.4.1 Become a Mode (-b)

The *sendmail* program can function in a number of different ways depending on which form the usage of -b argument. One form, for example, causes *sendmail* to display the contents of the queue. Another causes *sendmail* to rebuild the *aliases* database.

2.5.4.2 Daemon Mode (-bd)

The *sendmail* program can run as a daemon in the background, listening for incoming mail from other machines. The *sendmail* program reads its configuration file only once, when it first starts as a daemon. It then continues to run forever, never reading the configuration file again. As a consequence, it will never see any changes to that configuration file.

When you change something in the *sendmail.cf* configuration file, you *always* need to kill and restart the *sendmail* daemon. But before you can kill the daemon, you need to know how to correctly restart it. The daemon is usually started like this:

```
/usr/lib/sendmail -bd -q1h
```

The -bd command-line switch specifies daemon mode. The -q switch tells *sendmail* how often to look in its queue to process pending mail. The -q1h switch says to process the queue at one (1) hour (h) intervals.

2.5.4.3 Verbose (-v)

The -v command-line switch tells *sendmail* to run in *verbose* mode. In that mode, *sendmail* prints a blow-by-blow [4] description of all the steps it takes in delivering a mail message.

In the SMTP conversation your local machine displays what it is saying to the remote host by preceding each line with >>> characters. The messages (replies) from the remote machine are displayed with leading numbers.

```
220-remote.Domain Sendmail 8.6.12/8.5 ready at Fri, 13 Dec 1996 06:36:12 -0800
220 ESMTP spoken here
```

Once your *sendmail* has connected to the remote machine, your *sendmail* waits for the other machine to initiate the conversation. The other machine says that it is ready by sending the number 220, a dash (to say that it has more to say), and its fully qualified hostname (the only required information). If the other machine is running *sendmail*, it also says the program name *sendmail* and the version. It also states that it is ready and gives its idea of the local date and time.

The second line also starts with a 220, but that 220 is not followed by a dash. A dash means that more lines will follow, and the absence of a dash means that this is the last line. The "ESMTP spoken here" means that the remote site understands the Extended SMTP protocol. If the other machine were running V8.7 or later ***sendmail***, the ESMTP would precede **Sendmail** in the first line, and there would be no second line.

2.5.4 The sendmail.cf File

2.5.5.1 Overview

The *sendmail.cf* file is read and parsed by *sendmail* every time *sendmail* starts. It contains information that is necessary for *sendmail* to run. It lists the locations of important files and specifies the default permissions for those files. It contains options that modify *sendmail*'s behavior. Most important, it contains rules and rule sets for rewriting addresses.

The *sendmail.cf* configuration file is line-oriented. A configuration command, composed of a single letter, begins each line. Each configuration command is followed by parameters that are specific to it. Blank lines and lines that begin with the # character are considered comments and are ignored. A line that begins with either a tab or a space character is a continuation of the preceding line. *sendmail* program does not generally deliver mail itself. Instead, it calls other programs to perform that delivery. The M command defines a mail delivery agent (a program that delivers the mail).

2.5.5.2 Macros

The ability to define a value once and then use it in many places makes maintaining your *sendmail.cf* file easier. The D *sendmail.cf* command defines a macro. A macro's name is either a single letter or curly-brace-enclosed multiple characters. It has text as a value. Once defined, that text can be referenced symbolically elsewhere:

DRmail.us.edu ← *a single letter*
D{REMOTE}mail.us.edu ← *multiple characters*

Here, R and {REMOTE} are macro names that have the string mail.us.edu as their values. Those values are accessed elsewhere in the *sendmail.cf* file with expressions such as $R and ${REMOTE}.

2.5.5.3 Rules

At the heart of the *sendmail.cf* file are sequences of rules that rewrite (transform) mail addresses from one form to another. This is necessary chiefly because addresses must conform to many differing standards. The R command is used to define a rewriting rule:

```
R$-     $@ $1 @ $R     user -> user @ remote
```

Mail addresses are compared to the rule on the left ($-). If they match that rule, they are rewritten on the basis of the rule on the right ($@ $1 $@ $R). The text at the far right is a comment (that doesn't require a leading #).

2.5.5.4 Rule Sets

Because rewriting may require several steps, rules are organized into sets, which can be thought of as subroutines. The S command begins a rule set:

S3

This particular S command begins rule set 3. Beginning with V8.7 *sendmail*, rule sets can be given symbolic names as well as numbers:

Shubset

This particular S command begins a rule set named hubset. Named rule sets are automatically assigned numbers by *sendmail*. All the R commands (rules) that follow an S command belong to that rule set. A rule set ends when another S command appears to define another rule set.

2.5.5.5 Class Macros

There are times when the single text value of a D command (macro definition) is not sufficient. Often, you will want to define a macro to have multiple values and view those values as elements in an array. The C command defines a class macro. A class macro is like an array in that it can hold many items. The name of a class is either a single letter or, beginning with V8.7, a curly-brace-enclosed multicharacter name:

CW localhost fontserver ← *a single letter*
C{MY_NAMES} localhost fontserver ← *multiple characters (beginning with V8.7)*

Here, each contains two items: localhost and fontserver. The value of a class macro is accessed with an expression such as $=W or $={MY_NAMES}.

2.5.5.6 File Class Macros

To make administration easier, it is often convenient to store long or volatile lists of values in a file. The F command in *sendmail.cf* defines a file class macro. It is just like the C command above, except that the array values are taken from a file:

FW/etc/mynames
F{MY_NAMES}/etc/mynames ← *multiple characters (beginning with V8.7)*

Here, the file class macros W and {MY_NAMES} obtain their values from the file */etc/mynames*. The file class macro can also take its list of values from the output of a program. That form looks like this:

FM|/bin/shownames
F{MY_NAMES}|/bin/shownames ← *multiple characters (beginning with V8.7)*

Here, *sendmail* runs the program /bin/shownames. The output of that program is appended to the class macro.

2.5.5.7 Options

Options tell the *sendmail* program many useful and necessary things. They specify the location of key files, set timeouts, and define how *sendmail* will act and how it will dispose of errors. They can be used to tune *sendmail* to meet your particular needs.

The O command is used to set *sendmail* options. An example of the option command looks like this:

OQ/var/spool/mqueue
O QueueDirectory= /var/spool/mqueue ← *beginning with V8.7*

2.5.5.8 Headers

Mail messages are composed of two parts: a header followed (after a blank line) by the body. The body may contain virtually anything. The header, on the other hand, contains lines of information that must strictly conform to certain standards.

The H command is used to specify which mail headers to include in a mail message and how each will look:
With the advent of MIME (Multipurpose Internet Mail Extensions), the message body can now be composed of many minimessages, each with its own MIME header and sub-body.

HReceived: $?sfrom $s $.by $j ($v/$Z)$?r with r. id i?u for u.; $b

2.5.5.9 Priority

Not all mail has the same priority. Mass mailings (to a mailing list, for example) should be transmitted after mail to individual users. The P command sets the beginning priority for a mail message. That priority is used to determine a message's order when the mail queue is processed.

Pjunk= -100

This particular P command tells *sendmail* that mail with a Precedence: header line of junk should be processed last.

2.5.5.10 Trusted Users

For some software (such as UUCP) to function correctly, it must be able to tell *sendmail* whom a mail message is from. This is necessary when that software runs as a different user identity (*uid*) than that specified in the From: line in the message header. The T *sendmail.cf* command lists those users that are *trusted* to override the From: address in a mail message. All other users have a warning included in the mail message header.
The T command was ignored from versions V8.1 through V8.6 and restored under V8.7. With V8.7 it is actually implemented as the class $=t.

Troot daemon uucp

This particular T *sendmail.cf* command says that there are three users who are to be considered trusted. They are *root* (who is a god under UNIX), *daemon* (*sendmail* usually runs as the pseudo-user *daemon*), and *uucp* (necessary for UUCP software to work properly).

2.5.5.11 Keyed Databases

Certain information, such as a list of UUCP hosts, is better maintained outside of the *sendmail.cf* file. External databases (called *keyed* databases) provide faster access to such information. Keyed databases were introduced with V8.6 and come in several forms, the nature and location of which are declared with the K configuration command:

Kuucp hash /etc/mail/uucphosts

This particular K command declares a database with the symbolic name uucp, with the type hash, located in */etc/mail/uucphosts.*

2.5.5.12 Environment variables

The *sendmail* program is ultraparanoid about security. One way to circumvent security with *suid* programs like *sendmail* is by running them with bogus environmental variables. To prevent such an end run, V8 *sendmail* erases all its environment variables when it starts. It then presets the values for a small set of variables (such as TZ and SYSTYPE). This small, safe environment is then passed to its delivery agents. Beginning with V8.7 *sendmail*, sites that wish to augment this list may do so with the E configuration command.

EPOSTGRESHOME=/home/postgres

Here, the environment variable POSTGRESHOME is assigned the value */home/postgres*. This allows programs to use the *postgres*(1) database to access information.

2.5.5 The sendmail.mc File

The sendmail community came up with an easier option to customize sendmail. There also exists a sendmail.mc file, made up of macros. The sendmail.mc is easier to customize and this can be done in a simple and efficient manner.

Linux, by default, comes with the m4 macro processor. The m4 macro processor converts the sendmail.mc file into the corresponding sendmail.cf file. This file is further processed whenever sendmail is called.

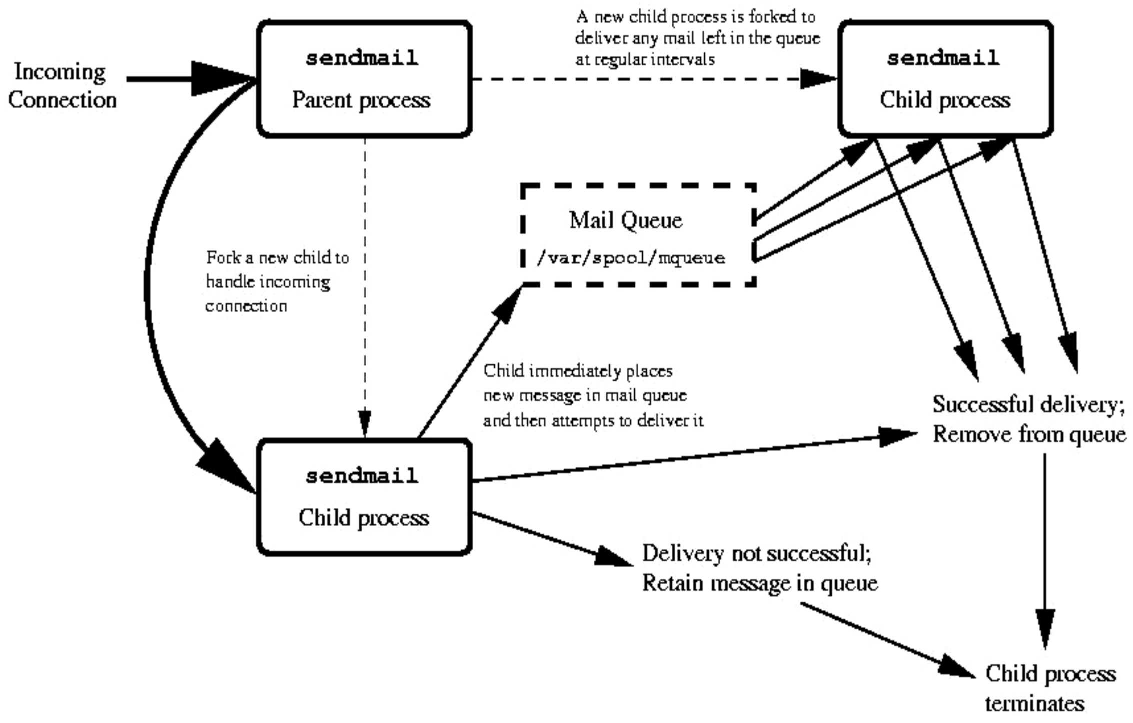
Sendmail uses the m4 macro processor to compile the configuration files. m4 is a macro processor, in the sense that it copies its input to the output, expanding macros as it goes. Macros are either built-in or user-defined, and can take any number of arguments. Besides just doing macro expansion, m4 has built-in functions for including named files, running UNIX commands, doing integer arithmetic, manipulating text in various ways, recursion, etc. m4 can be used either as a front-end to a compiler, or as a macro processor in its own right.

We can insert our own macros in the appropriate places in the sendmail.mc file. These shell scripts subsequently invoke our filter. The filter processes the content of the mail and blocks mail that contain sensitive information.

2.5.7 The Sendmail MTA

In a normal configuration, sendmail sits in the background waiting for new messages. When a new connection arrives, a child process is invoked to handle the connection, while the parent process goes back to listening for new connections.
When a message is received, the sendmail child process puts it into the mail queue (usually stored in /var/spool/mqueue). If it is immediately deliverable, it is delivered and removed from the queue. If it is not immediately deliverable, it will be left in the queue and the process will terminate. Messages left in the queue will stay there until the next time the queue is

processed. The parent sendmail will usually fork a child process to attempt to deliver anything left in the queue at regular intervals.


Incoming Connection
sendmail
Parent process
A new child process is forked to deliver any mail left in the queue at regular intervals
sendmail
Child process
Mail Queue
/var/spool/mqueue
Fork a new child to handle incoming connection
Child immediately places new message in mail queue and then attempts to deliver it
Successful delivery; Remove from queue
sendmail
Child process
Delivery not successful; Retain message in queue
Child process terminates

2.5.7.1 Step-by-Step Analysis

- Sendmail Daemon

– Listens on port 25
– Accepts connection
– Forks child process
– Hands off connection
– Goes back to listening to port 25

Alternatively, if the load average is too high, it will go to sleep for fifteen seconds and check the load average.

- Child Process

– Listens on port 25
– Accepts connection
– Forks child process
– Hands off connection
– Goes back to listening to port 25

Alternatively, if the load average is too high, it will go to sleep for fifteen seconds and check the load average.

- Grand-child Process

  – Waits for “Hello” or “EHLO” command
  – Waits for “MAIL From:” command
  – Creates queue files in /var/spool/mqueue
  – Waits for “RCPT To:” command(s)
  – Waits for “DATA” command
  – Waits for “.” (on a line by itself)
  – Writes the message to the mqueue
  – Makes initial delivery attempt, and quits

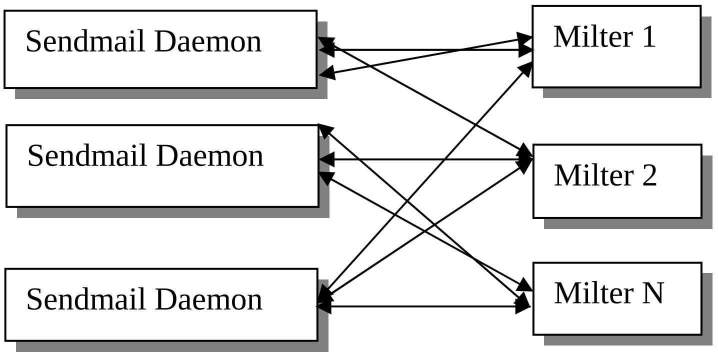
- Transmitting a Message

  – Separate queue-runner process(es) started up periodically by daemon, cron, or manually
  – Scans entire queue directory
  – Calculates priority of each message
  – Makes delivery attempt for each message, starting with highest priority, each recipient domain handled in turn

2.5.8 Filtering Mail with Milter

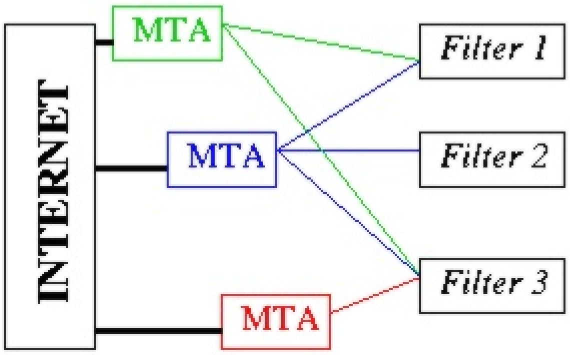
A mail filter program that allowes us to process a mail according to our requirement. For example: we can develop a filter program to check the content of the mail body for sensitive information or to classify and deliver a mail to the different folders we maintain in our mail box (what we do with our yahoo mail account). Milter is Sendmail's mail Filter API and provides third-party programs to access mail messages as they are being processed by the Mail Transfer Agent (MTA), allowing them to examine and modify message content and meta-information. Milter (Mail Filter) is both a *protocol* and a *library*. Milter was designed by the Sendmail development team, and has been part of Sendmail since 8.10. _. Milter lets filters “listen in” to the SMTP conversation and modify SMTP responses. Milter is extremely powerful and can let you do amazing things with e-mail. Filters can process messages' connection (IP) information, envelope protocol elements, message headers, and/or message body contents, and modify a message's recipients, headers, and body. The MTA configuration file specifies which filters are to be applied, and in what order, allowing an administrator to combine multiple independently-developed filters.

2.5.8.1 Milter Architecture

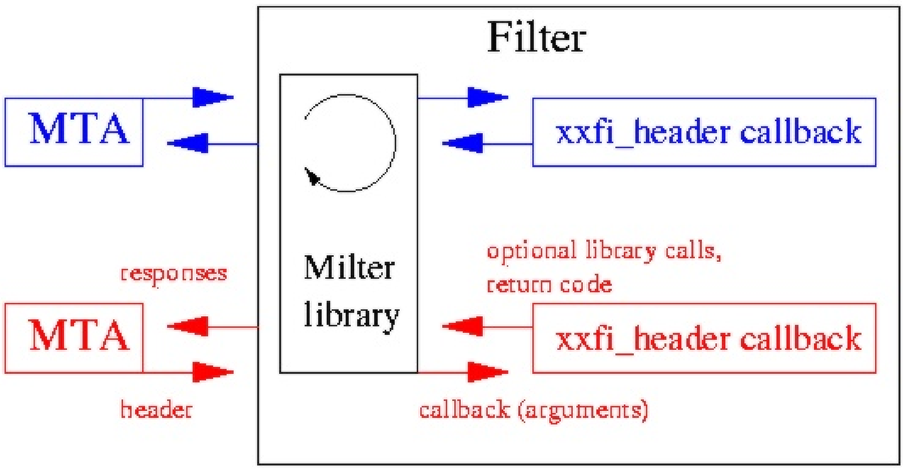


Sendmail is single threaded but forks into multiple processes. Milter library is multi- threaded and a given sendmail installation can have multiple mail filters.

Filters run as separate processes, outside of the sendmail address space. Each filter may communicate with multiple MTAs at the same time over local or remote connections, using multiple threads of execution. Figure illustrates a possible network of communication channels between a site's filters, its MTAs, and other MTAs on the network:

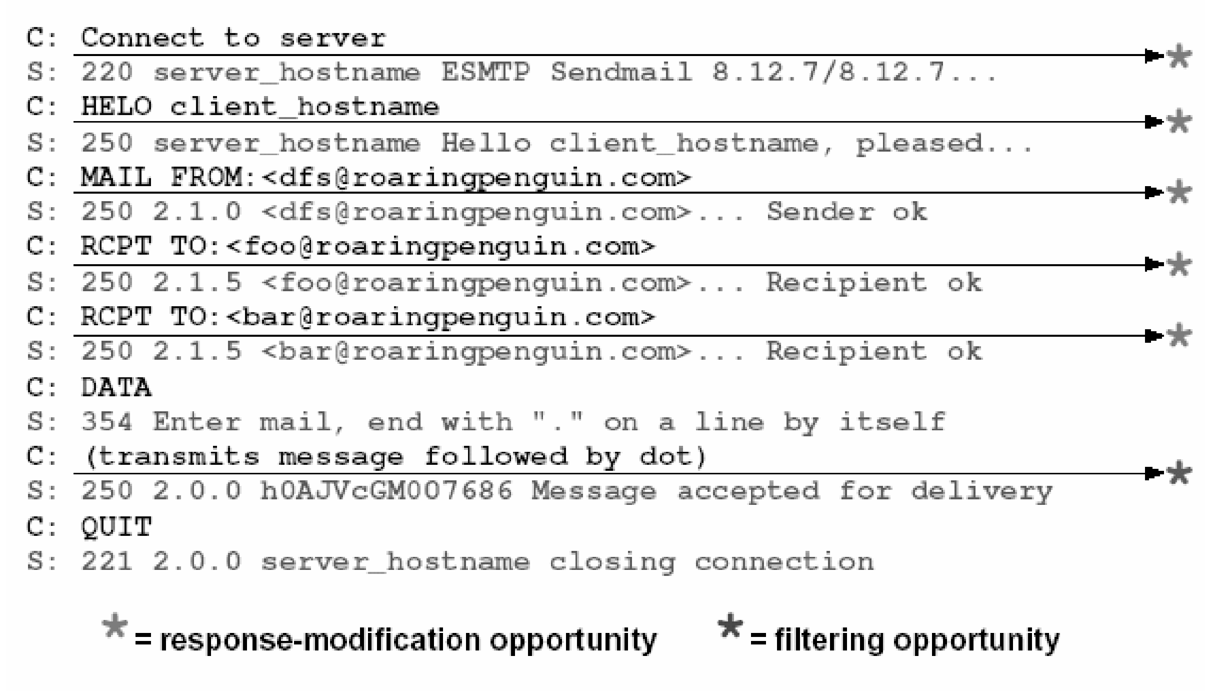


The Milter library (libmilter) implements the communication protocol. It accepts connections from various MTAs, passes the relevant data to the filter through callbacks, then makes appropriate responses based on return codes. A filter may also send data to the MTA as a result of library calls. Figure shows a single filter process processing messages from two MTAs.



2.5.8.2 Milter API Operation

At various stages of the SMTP conversation, Sendmail sends a message over the socket to the milter. The milter library invokes a *callback* into the filter code. It then sends a reply message back to Sendmail containing the return value from the callback. In addition, you can call milter library functions that send special messages to Sendmail to modify aspects of the message.



```
C: Connect to server
S: 220 server_hostname ESMTP Sendmail 8.12.7/8.12.7...
C: HELO client_hostname
S: 250 server_hostname Hello client_hostname, pleased...
C: MAIL FROM:<dfs@roaringpenguin.com>
S: 250 2.1.0 <dfs@roaringpenguin.com>... Sender ok
C: RCPT TO:<foo@roaringpenguin.com>
S: 250 2.1.5 <foo@roaringpenguin.com>... Recipient ok
C: RCPT TO:<bar@roaringpenguin.com>
S: 250 2.1.5 <bar@roaringpenguin.com>... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: (transmits message followed by dot)
S: 250 2.0.0 h0AJVcGM007686 Message accepted for delivery
C: QUIT
S: 221 2.0.0 server_hostname closing connection
```

* = response-modification opportunity
* = filtering opportunity

Typical SMTP Conversation with Milter

2.5.8.3 Milter API Functions

2.5.8.3.1 Initialization:

You “register” a filter and tell what kinds of callbacks you're interested in, and whether or not you might modify the message headers and body.

smfi_register	Register a filter
smfi_setconn	Specify socket
smfi_settimeout	Set timeout
smfi_main	Enter main milter loop

2.5.8.3.2 Data Access:

Accessing Sendmail macros lets you access a lot of useful info. Private data useful for storing thread-specific data can be accessed. We can alsi set SMTP reply to anything you like.

smfi_getsymval	Get value of a Sendmail macro
smfi_getpriv	Get arbitrary private data
smfi_setpriv	Set arbitrary private data
smfi_setreply	Set SMTP reply code

2.5.8.3.3 Message Modification:

Recipient functions affect *envelope recipients*, not headers.

smfi_addheader	Add a header
smfi_chgheader	Change a header
smfi_addrcpt	Add a recipient
smfi_delrcpt	Delete a recipient
smfi_replacebody	Replace message body

2.5.8.3.4 Milter API: Callbacks

xxfi_connect	Called when connection made
xxfi_helo	Called after HELO
xxfi_envfrom	Called after MAIL FROM:
xxfi_envrcpt	Called after RCPT TO:
xxfi_header	Called for each header
xxfi_eoh	Called at end of all headers
xxfi_body	Called for each “body block”
xxfi_eom	Called at end of message
xxfi_abort	Called if message aborted
xxfi_close	Called when connection closed

2.5.8.3.5 Callback Types

Connection oriented: Apply to connection as a whole. Eg: *xxfi_connect*, *xxfi_helo*, *xxfi_close*.
Recipient oriented: Apply to a single recipient only. Eg: *xxfi_envrcpt*
Message-oriented: Apply to a message. All other callback types are message-oriented.

2.5.8.3.6 Callback Return Values

SMFIS_CONTINUE: Continue processing
SMFIS_REJECT: For connection-oriented callback: close the whole connection.
For message-oriented: reject the message.
For recipient-oriented: reject only this recipient.
SMFIS_DISCARD: For message- or recipient oriented routine: silently discard message.
SMFIS_ACCEPT: Accept without further filtering.
SMFIS_TEMPFAIL: Return a temporary-failure.

3. SYSTEM ANALYSIS AND REQUIREMENTS

3.1 Problem definition

The objective of the project is to filter the content of the mail that goes out of the NAL mail server.

Every mail that goes out of the mail server is scrutinized. The sender address and the user-name are identified. The subject of the mail is scanned for words that are considered to be sensitive for that particular employee. Then the body of the mail is parsed. The attachments, if any, are identified and classified. Based on the type of the attachment file, its content is parsed.

At any juncture, if the mail is found to contain a word that is considered to be sensitive for that particular employee of the organization, the mail is blocked. A message is sent to the employee in question informing him that the mail was blocked and could not be sent. Besides this, a copy of the mail is also sent to the administrator informing him of the breach of confidence.

3.2 Existing System

National Aerospace Laboratories is presently being served by Linux 7.3 that is used to run the SMTP server. The MTA used is a very popular package called Sendmail. NAL uses Sendmail version 8.11.6 to serve the employees. The NAL website www.nal.res.in also provides the employees with a web client interface to send and receive mails.

The present scenario is such. There exists no mechanism on the server that will scan the content of the mail or the attachments sent along with the mail. Any message and any type of attachments can be sent out of the organization. This makes the system vulnerable to leakage of information inadvertently or otherwise.

3.3 Proposed System

The system to be developed will have the following features. The main aim of the proposed system is to prevent the outflow of sensitive information from an institution like the National Aerospace Laboratories, which may result in unprecedented consequences. The privacy of the information of the organization is a matter of great concern.

The proposed system is a filter that will be incorporated into the mail server. The devised filter will be run before the MTA is handed the mail for further processing and transferring. The filter goes through the mail thoroughly and checks each section for any word that may be considered sensitive and hence inappropriate to use.

Every mail that is sent by any employee from his NAL mail server account is scrutinized. The information that is sent along with the mail, in the form of the headers, is also scrutinized. From this the identity of the person who is sending the mail is acquired. Once we have the name of the employee, we retrieve the file corresponding to that particular employee. This file will contain the list of words that the employee is not authorized to use.

Now the mail is parsed for these words, section after section. The subject of the mail is first checked. Then the actual mail body is parsed. The attachments, if any, are considered separately. If the attachment is of a type that does not confirm to the policy of laboratory, the mail is immediately blocked. Else the attachment is processed according to the type of attachment file. The content is parsed for sensitive keywords.

If at any point of processing, if the mail is found to contain any sensitive word that the employee is not authorized to use, the mail is immediately blocked. A warning message is sent back to the employee's account informing him that the mail has not been relayed. To prevent further occurrence of the same, the administrator is also sent a copy of the mail. This gives the administrator room to take appropriate action.

3.4 Software Requirement Specification

3.4.1 Introduction

3.4.1.1 Purpose of this Document

This Software Requirement Specification describes the function and performance requirements of a Message Filter used to scan through the body of an email. The filter is intended to search the body content for unauthorized use of sensitive keywords and block mails that contain the same.

3.4.1.2 Scope of the Development Project

The project aims at filtering emails that go out of the *National Aerospace Laboratories (NAL)* mail server.

Securing sensitive research information is a key concern for organizations like the National Aerospace Laboratories. Filtering the emails that are sent out through the NAL server helps serve this purpose. This keeps track of not only the information that is sent out but also the personnel that are involved in the exercise.

A Mail User Agent (MUA) is any program that users run to read, reply to, compose and dispose of email. A Mail Transfer Agent (MTA) is a highly specialized program that delivers mail and transports it between machines. "Sendmail" is the most popular MTA in the Linux platform. Almost every client program (Mail User Agents) sends emails using Sendmail. Sendmail is the MTA used to transfer the mail going through the NAL mail server. Our objective is to insert our filter into the appropriate places in the "sendmail" MTA.

The filter has to check for sensitive information. The filter scans the body of the email for some pre-defined keywords. These keywords are maintained in a database/ file. For each keyword, we have a set of mail users who are authorized to use the keyword. If the user is not permitted to use that particular word / phrase, the mail is blocked and the user is notified by sending a mail to his mail account.

3.4.1.3 Overview of Document

The SRS is organized as follows. First we consider the general description of the software package. Here we review the characteristics of the user of this package. We then consider the perspective of the product. This is followed by a brief description of the functional and data requirements of the system. We also review the assumptions made while developing the package. The constraints of the system are also mentioned.

The functional requirements of the package are considered in detail. The different requirements are analyzed and considered.

3.4.2 General Description

3.4.2.1 User Characteristics

This application can be used by system administrators of organizations who find the need to keep a check on the information that flows out of the organization through its employees. The system administrator has access to the database/ file and it is the responsibility of the administrator to update the information present in the database/ file. The sensitive keywords present in the database/ file can be customized as per the requirements and the policies of the institutions using the package. The administrator is expected to have the basic knowledge of Sendmail and the mail server.

3.4.2.2 Product Perspective

The product resides on the NAL mail server. The filter is called by the MTA after the mail is passed to it by the MUA. The action of the filter takes place before the MTA accesses the mail.
NAL uses Sendmail as its default MTA. The Sendmail package is the most popular MTA in the Linux platform. The filter is incorporated into the larger package, Sendmail. Support for the filter is available on Sendmail versions 8.10.x and higher.

3.4.2.3 Overview of Functional Requirements

The purpose of the project is to develop a filter that scans the body as well as the attachments of any mail that goes out of the NAL mail server.

When a user of the NAL mail server compiles a mail and chooses to send it, the mail is transferred from the MUA to the MTA. Our filter is now invoked using Mail-Filter (Milter) APIs. The filter scans the content of the body of the mail. It also searches for the presence of attachments. If an attachment is present, then it would be encoded using the various MIME (**M**ultipurpose **I**nternet **M**ail **E**xtension) encoding techniques. The attachment is decoded and scanned for keywords. The mails found to contain these keywords are checked for authorization. If the person who sent the mail is not authorized to use the sensitive keyword, then the mail is not sent. Instead, a mail is sent to the offender’s mail account informing that the mail was not sent to the intended receiver.

3.4.2.4 Overview of Data Requirements

The input for the above product is an email sent by any user of the NAL mail server. Every mail is sent, the filter is called. The mail is intercepted and the filter acts on the mail. If it contains sensitive information, a mail is sent to the user informing him about the non-delivery of the mail.

3.4.2.5 Constraints and Assumptions

The following assumptions are made while developing the package.

- The mail is not encrypted in any manner or the data is not in a language other than English.

The constraint of the package is:

- The package works only with Sendmail 8.10.x and higher versions.

3.4.3 Specific Requirements

3.4.3.1 Detailed Description of Functional Requirements

Sendmail is the most popular MTA that is used in the Linux platform. Sendmail was the first program designed to route electronic mail between networks, developed in 1981 by Eric Allman at UC Berkeley. Developed and distributed as an open source product, it has been widely deployed and tested in a huge number of large-scale e-mail environments, including many of the world's largest ISP's.

Sendmail is the program that acts like a traffic cop in routing and delivering mail on Unix-based networks. It is a Mail Transport Agent (MTA), accepting mail from Mail User Agents (MUAs), mail users (humans), and other MTAs. Then, it delivers that mail to Mail Delivery Agents (MDAs) on the local machine, or transports that mail to another MTA on another machine.

- ❖ When Sendmail receives the mail, the filter program is called. The mail may contain attachment files. If the mail is devoid of attachments, the body of the mail is parsed.

- ❖ The sensitive keywords are retrieved from the database. If a sensitive keyword is found, the authorization of the person under question is checked. If it is found that he/she is authorized to use the given word, the mail passes through the filter and handed over to the MDA.

- ❖ If the person is not allowed to use the keyword, the mail does not pass through the filter. Instead, a mailer daemon is sent to the offender's mailbox informing him/her that the particular mail was not delivered to the expected receiver.

- ❖ If an attachment is sent along with the mail, then the attachment of the mail is first identified and file type is checked.

- ❖ If the attachment file type is found to be objectionable (which will depend on the organizational policies) then the mail is blocked . In other cases assuming that the attachments are text files, content of each of them is parsed for the sensitive keywords and the mail is dealt with as described above.

- ❖ In case of compressed files, the file is first uncompressed and it subjected to similar procedures described above.

3.4.3.2 Detailed Description of Technical Requirements

In a normal configuration, Sendmail sits in the background waiting for new messages. When a new connection arrives, a child process is invoked to handle the connection, while the parent process goes back to listening for new connections. When a message is received, the sendmail child process puts it into the mail queue (usually stored in /var/spool/clientmqueue).

If it is immediately deliverable, it is delivered and removed from the queue. If it is not immediately deliverable, it will be left in the queue and the process will terminate. Messages left in the queue will stay there until the next time the queue is processed. The parent Sendmail will usually fork a child process to attempt to deliver anything left in the queue at regular intervals.

- To achieve the required goals the filtering activity has to be invoked each time Sendmail opens a SMTP connection for delivering the mails to its appropriate destination.

- The filter has to differentiate between outbound and inbound mails and only process the outbound mails. **The incoming mails are not subjected to any kind of filtering activity.**

- The retrieval of the unauthorized keywords has to be done on the basis of the "From" field entry in the message header which gives the sender's name, although the list of sensitive words is common to all.

- The outbound mail which does not have any attachment will fail to pass through the filter in the following cases:

  - The "subject" of the message header contains any of the keyword.

  - The body part of the message after parsing is found to have any of the keywords.

  In any of the above cases a mail has to be sent back to the sender informing him about the rejection of the delivery of the message. This is accomplished as follows. The 'To:' address is replaced with the 'From:' address and the body replaced with a warning message.

- In case the mail contains attachments following steps are to be followed

  - The attachment is encoded using some standards like Base64 or Uuencode before sending it over the network. This has to be first decoded.

  - Once the attachment is decoded it is checked for file type. If file type of any attachment is not allowed by the organization then the mail will be rejected and will be dealt with as described above.

  - If some text file or MS-word file is found then content of those files have to be parsed and actions has to be taken accordingly.

  - The attachment may be compressed to give .zip or .tar or related formats. In that case the attachment file has to be uncompressed recursively so that we get all the files, which were subjected to compression. After getting all the files they are treated in exactly similar manner as other attachment files.

3.4.3.3 Detailed Description of Interface Requirements

3.4.3.3.1 User Interface Requirements

The potential user who is involved in the usage or manipulation of the software is the system administrator. Hence he should be provided with the facilities for the customization of this filtering procedure.

- ❖ Provision should be there so that the administrator can maintain well designed and easily maintainable database of authorized keywords for every client of the mail sever.
- ❖ The administrator should add or delete keyword in the database depending on the situations.
- ❖ Keywords in the database should not be redundant.
- ❖ Running of the filtering software should be synchronized with Sendmail daemon. This will avoid manual initiation of the filter by the administrator.

3.4.3.3.2 Data Interface Requirements

- ❖ The data which is handled by filter software is precisely the mails which are handed over to Sendmail for delivery.
- ❖ Sendmail works on SMTP. So to get the required data for the software it is necessary to keep track of the SMTP connection and manipulate the mails before Sendmail processes them.
- ❖ The software should be able to interpret the data which is coming through the SMTP connection. It should be well equipped to handle mails in RFC 822 format as well as MIME formats.

3.4.3.4 Attributes and Consideration

3.4.3.4.1 Availability

The filter software should always run along with Sendmail in the mail server. So as long as mail server is activated filter should also be running in the mail server.

3.4.3.4.2 Security

The present mail-system has following security lapses.

- ➢ The existing system does not scrutinize the content of the mail, so sensitive information regarding the different research and development projects of in the laboratory can be exposed to the outside world via emails using NAL mail server.

- ➢ Besides these users having accounts in NAL mail-server are expected to use their mail account mainly for official purpose. So the language used should be fairly professional, polite and dignified. But in the present set up no restriction has been put on the mail content. So a mail containing indecent or abusive words will also be forwarded to its recipient. This could be very bad for the institution’s reputation.

Introduction of the software in the present system should remove these vulnerabilities from the system.

- The software will allow only the authorized persons to write about the keywords that are considered to be sensitive for the organization. Thus a person is allowed to write about a particular field of research only if he is a responsible member associated with it.
- The policy that has to be followed to address the second problem is that any occurrence of abusive words in the mail will subject it to be rejected by the filter software.

3.4.3.4.3 Maintainability

Considering the aspect of maintainability the filtering software has to follow few criteria

- It should have good design so that in case some error is reported *corrective maintenance* will just be a matter of change of code which is not at all an expensive procedure. In case of this software the design should be such that the different parts of the email(i.e. header, body, attachments etc) are handled by different functions. These functions should have sub-functions that will handle the actions to be taken concerning that part of the email. The functions should be independent of each other so that in case any error is reported changing only that particular function serves the purpose and these changes do not affect other functions.
- It should be *adaptive* enough to be suited in different environment with a very little modification. As example if the MUA is changed from "pine" to "mutt" the software should work with little changes.

- It should be suitable for *perfective maintenance* i.e. it should be able to incorporate new functionalities as the organizational or business requirement changes. The filtering software should be customizable according to the policies of an organization. As an example if the list of sensitive keywords for which a particular account holder is authorized needs to be changed or the list of allowed attachment file-types needs to be changed, the software should give facility to do so and work accordingly.

3.5 Hardware Requirements

- Processor : Pentium I higher or its equivalent
- Ram : 64MB or more
- Secondary Memory : 2 GB or above
- External Devices : Network Interface Card

3.6 Software Requirements

- Operating System : BSD UNIX or its variants or Linux (Kernel 2.4.x or above)
- Mail Server : Sendmail 8.12.5 or above with Milter Api (Sendmail Filter Api) support
- Library Support:
    - GNU C Libraries
    - Milter Library (mfapi.h, libmilter.a, libsm.a)
    - GNOME Library (gnome.h)
    - GTK+ and GDK Libraries (gdk.h and gtk.h)

4. SYSTEM DESIGN

4.1 Feasible Solutions

The filter that does the message filtering has to be incorporated into the mail system. There are a number of options available to do the same. Let us consider each of them in detail.

4.1.1 Processing the mail in the queue

This is the first option that was considered. Every mail that goes through the Sendmail MTA passes through a queue. This is called *clientmqueue* in the Sendmail 8.10.x versions. The entire mail, including the mail and the associated headers will be present when the mail is waiting in the queue for clearance to the next level.

The mail is captured at the juncture and processed. The filter is run on the mail body and the attachments. The mail is subsequently released from the queue, for further processing.

4.1.2 Altering the Sendmail source code

In this option, we can alter the source code of sendmail itself. Sendmail is a highly complex program that has vast application. The sendmail program has to transport mail between a wide variety of machines.

We can attempt to alter the code of sendmail to accommodate our filter, which is run every time a mail is sent. But the code of sendmail is highly complex and very rarely tampered with, even by the sendmail experts. Hence this is a highly complex and risky option.

4.1.3 Altering the sendmail configuration file

The sendmail program is composed of several parts, including programs, files, directories and the services it provides. Its foundation is a configuration file that defines the location and behavior of these other parts. The configuration file of sendmail is sendmail.cf. This can be found in one of the directories /etc, /usr/lib or /etc/mail. It provides all the central information that controls the sendmail program’s behavior. The configuration file is read and parsed by sendmail every time it starts up.

Because sendmail is invoked for every email sent, its configuration file is designed to be easy for sendmail to parse rather than easy for humans to read. The syntax of sendmail.cf file is highly cryptic. Altering this code is a very tedious process and requires profound knowledge of the syntax and the details of working of sendmail.

4.1.4 Altering the sendmail.mc file

The sendmail community came up with an easier option to customize sendmail. There also exists a sendmail.mc file, made up of macros. The sendmail.mc is easier to customize and this can be done in a simple and efficient manner.

Linux, by default, comes with the m4 macro processor. The m4 macro processor converts the sendmail.mc file into the corresponding sendmail.cf file. This file is further processed whenever sendmail is called.

Sendmail uses the m4 macro processor to compile the configuration files. m4 is a macro processor, in the sense that it copies its input to the output, expanding macros as it goes. Macros are either built-in or user-defined, and can take any number of arguments. Besides just doing macro expansion, m4 has built-in functions for including named files, running UNIX commands, doing integer arithmetic, manipulating text in various ways, recursion, etc.

We can insert our own macros in the appropriate places in the sendmail.mc file. These shell scripts subsequently invoke our filter. The filter processes the content of the mail and blocks mail that contain sensitive information.

4.1.5 Using Mail Filter APIs

Sendmail also provides for mail filters, called milters. Milter (Mail fILTER) is both a protocol and a library. Milter was designed by the Sendmail development team, and has been part of Sendmail since 8.10. Milter lets filters “listen in” to the SMTP conversation and modify SMTP responses. Milter is extremely powerful and can let you do amazing things with e-mail.

Sendmail's Content Management API (milter) provides third-party programs to access mail messages as they are being processed by the Mail Transfer Agent (MTA), allowing them to examine and modify message content and meta-information. Filtering policies implemented by Milter-conformant filters may then be centrally configured and composed in the MTA configuration file.

The filter that is to process the mails can be incorporated in the Milter to parse the sections of the mail and even alter if the need arises.

4.2 Adopted Solution

Considering all the options available to integrate our customized filter into sendmail, we arrived at the conclusion that using Milter APIs would be the most appropriate choice.

The milter basically has the following function. Every time a mail is sent, the MTA calls the milter code before the action of MTA takes place. The code opens a socket to listen to incoming mail. Once a mail is received, each header and the body of the mail can be processed. It is also possible to set an option that dictates that the received mail should not be sent if the milter code is not found.

The Sendmail Content Management API (Milter) provides an interface for third-party software to validate and modify messages as they pass through the mail transport system. Filters can process messages' connection (IP) information, envelope protocol elements, message headers, and/or message body contents, and modify a message's recipients, headers, and body. The MTA configuration file specifies which filters are to be applied, and in what order, allowing an administrator to combine multiple independently-developed filters.

4.2.1 Advantages of using Milter

- Simplicity - Milter provides various functions that help access the different sections of the mail thus simplifying the job of the filter writer.
- Performance - Simple filters do not seriously impact overall MTA performance

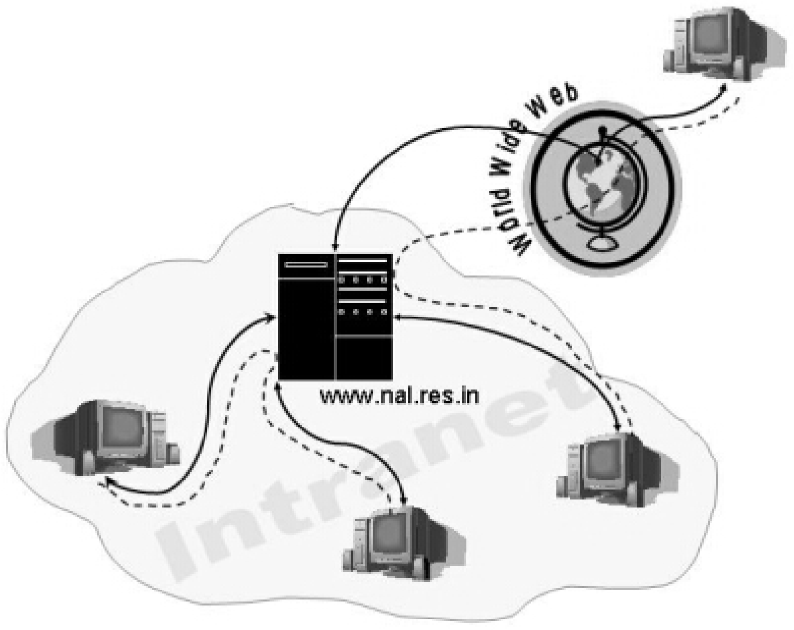
- Filtering at the server is the best place to filter mail and Send mail's Milter API helps achieve this.

The above reasons substantiate the decision to adopt Milter and the Milter APIs to incorporate our filter into the mail service system.

4.3 Overall System Description

4.3.1 Overview of the present set up of the mail-server

The mail-server setup in the National Aerospace Laboratory can be visualized in the following diagram:



The employees of NAL are provided with a email ID. These use accounts resides on the Linux server machine as shown in the diagram. A user can avail the mailing facility by two ways:

- ❖ If he is in the Intranet he can connect to his account by using telnet and send or receive mail using any of the user agents which can be activated from command line (like pine, mail, mutt).
- ❖ If he is in the internet or outside it he can make use of the webmail facility which is provided by NAL website.

4.3.2 Different components used in the present system

The three major three components of the mail-server are

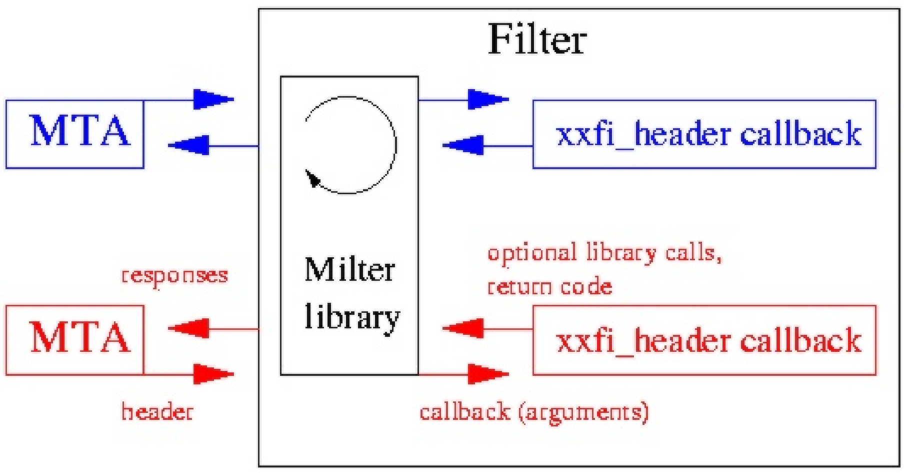
- ❖ MUA(mail user agent)
- ❖ MTA(mail transfer agent)
- ❖ MDA(mail delivery agent)

In the present set-up the software used for these three parts are

- ❖ PINE: It is used as a MUA. The name as an acronym of the term **p**rogram for **I**nternet **n**ews and **e**-mail.
- ❖ SENDMAIL: It is used as the MTA.
- ❖ PROCMAIL: It is used as the MDA.

4.3.3 The component where the mail filtering software resides

- The mail filtering software works in synchronization with the MTA i.e. mail transfer agent. In the present set-up it works along with Sendmail (MTA).
- Whenever a SMTP connection is opened by Sendmail for the transfer of the mail the byte-streams pass through the filtering software. The content of the mail is scanned during that time.



4.3.4 Brief description of the working of the mail-filter

As soon as a SMTP connection is opened by *sendmail* for mail delivery the byte-stream passes through the mail-filter. The different parts of the mail are processed by the different callbacks of Milter.

The step by step processing is given below

- While processing the header part of the mail, the value of the “From” field is determined and on the basis of that it is determined whether the mail is outbound or inbound. The mail is said to be outbound only when the “From” address is nothing but NAL’s domain address.
- The mail is processed further only if it is outbound. Else it is allowed to go to its destination.
- Next the subject of the mail is checked. If any keyword is found for which the the sender is not authorized, the mail will not be delivered to its destination instead it

will be sent back to the sender's address. Same policy is followed if any abusive word is found in the content of the subject.

- If subject is not found to be objectionable the body part is checked and same rule described above is followed.

- If the mail contains attachments then whole mail is given to "ripmime" which will separate the attachment files after decoding them (the attachment comes encoded by different encoding techniques such as BASE64).

- If the file type of the attachments is considered permissible based on the policies of the organization then each of them is processed individually and content is checked in the same way.

4.3.5 Design of the database maintained

There are three aspects which are being checked in each mail for deciding whether it is allowable or not. They are-

- Whether the sender is using a keyword for which he is not authorized.
- Whether the sender is using any abusive word.
- Whether the file type of the attachment is permissible by the organization.

For checking these three aspects a general list of sensitive words based on the projects handled by NAL (say list A) is maintained. Another list of sensitive words for employees which they are authorized to use based on their group (say list B) is also maintained. Now the list of maintained unauthorized sensitive words for any employees obtained from A-B. Along with this a list of abusive words and allowed file types are also maintained.

The different options available for storing the data were-

1. ***Using any relational database software :*** There are quite a few relational database software which open source and can be used in Linux platform such as

   - mysql
   - postGres

2. ***Using files:*** Files can be maintained which will contain the records in a systematic way.

Between these two options, in this particular scenario **files** are preferable to **relational database software** for the following reasons-

- A mail-server has to handle large number of mails. The load on the mail-server could be quite heavy. So the mail-filter software has to work very fast. Time is probably the most valuable resource being handled in this particular system. Undoubtedly using **file** is much faster than using a **relational database**.

- Unix system provides extensible facility for manipulation of strings or content of file. This makes the retrieval of data from the file very convenient.

- Besides these it is a very common practice in Unix systems to maintain records in files. As for an example- the information about user accounts is kept in the file /etc/passwd.

Considering the above mentioned advantages **files** are used for storing data for the software.

Organization of files

Three types of text files are used as a ***static record*** in the present system.

1. The *universal* files containing
    - every possible keyword which are treated as sensitive
    - all the abusive words.
2. The files containing the sensitive keywords for which members of a group are authorized.
    - These files are maintained against each user group.
    - An user is authorized to use all the keywords which are enlisted in the files corresponding to the groups in which the user is a member.
    - The groups are enlisted in the file /etc/group which is a part of the UNIX system. In this file, user groups as well as system groups are enlisted. There is no particular way to differentiate between them. But as a convention, the user group ids in /etc/group are given values above 500. This particular characteristic is used to differentiate between these two types of groups (system groups and user groups). A file is created for each of these user groups which contain the sensitive keywords the group members are authorized to use.
3. The file containing all *permissible file types* for attachment.

The ***dynamic*** file containing the list of keywords which a user is not authorized to use, is generated from the first two types of files in the following way-

- In the /etc/group file, corresponding to each group the name of the members are given. Using this entry we retrieve all the group names in which the user is a member.
- After this the content of the files corresponding to these groups are concatenated in a single *temporary* file which gives all the allowed keywords for the particular user.
- This *temporary* file is compared with the *universal* file which contains all the keywords and the words which are unique in the *universal* file are found.
- These entries are written in a different file which gives list of keywords which the user is not authorized to use.

4.5 User Interface Design

The mail-filter requires a GUI interface called the Sfilter Configuration Tool to help the system administrator configure the filter.

4.5.1 Sfilter Configuration Tool

The Configuration Tool provides the following features to the system administrator:

- ♦ A single column display of all the keywords in the universal list of sensitive words
- ♦ A two column display of all the user groups and the list of authorized sensitive words corresponding to each group.
- ♦ The administrator can add or delete words from the columns displaying sensitive words.
- ♦ The administrator is provided with a text box into which he can key in new words.

  On clicking the Add to Sensitive List button, the keyword will be added to the Universal Sensitive List. Checking is done to ensure that duplicates are not added.

  On clicking the Add to Group button the keyed in keyword is added to the list corresponding to the selected group. The word is compared with the universal list and the group list to ensure that there is no duplication. In the event that the word is a totally new word, then it is added to both the lists.
- ♦ A separate window containing a column display of all words regarded as abusive is also provided to the administrator, on the click of a button. He/ She can add new words or delete new words from this list.
- ♦ A separate window that displays a list of all valid domains(for emails) within the organization is also provided to the user (system administrator). This list can be modified as the need arises. For example, a new domain that provides addresses such as xyz@css1.nal.res.in is setup, then the administrator needs to tell Sfilter that this is valid domain, otherwise Sfilter will mails from this domain (considers all such mails have outgoing)

4.6 Design constraints

- ▪ The software works with the basic assumption that the content of the mail is *not encrypted.*
- ▪ The language used should only be *plain English*.
- ▪ The software searches only for the keywords in the content of the mail. It *does not consider the context* in which the keywords are used. A sensitive keyword can occur in the mail in a very harmless context which does not reveal any information about it. The present software does not differentiate them.

To do so the software needs to interpret the language based on the context of usage. There could have been two probable ways of doing it-

1. Making a finite machine which works on the basis of a grammar. Setting up some semantic meaning with each rules of grammar and thus interpret the language.
2. Making an intelligent system by the use of neural network/artificial intelligence which will be able to understand the language.

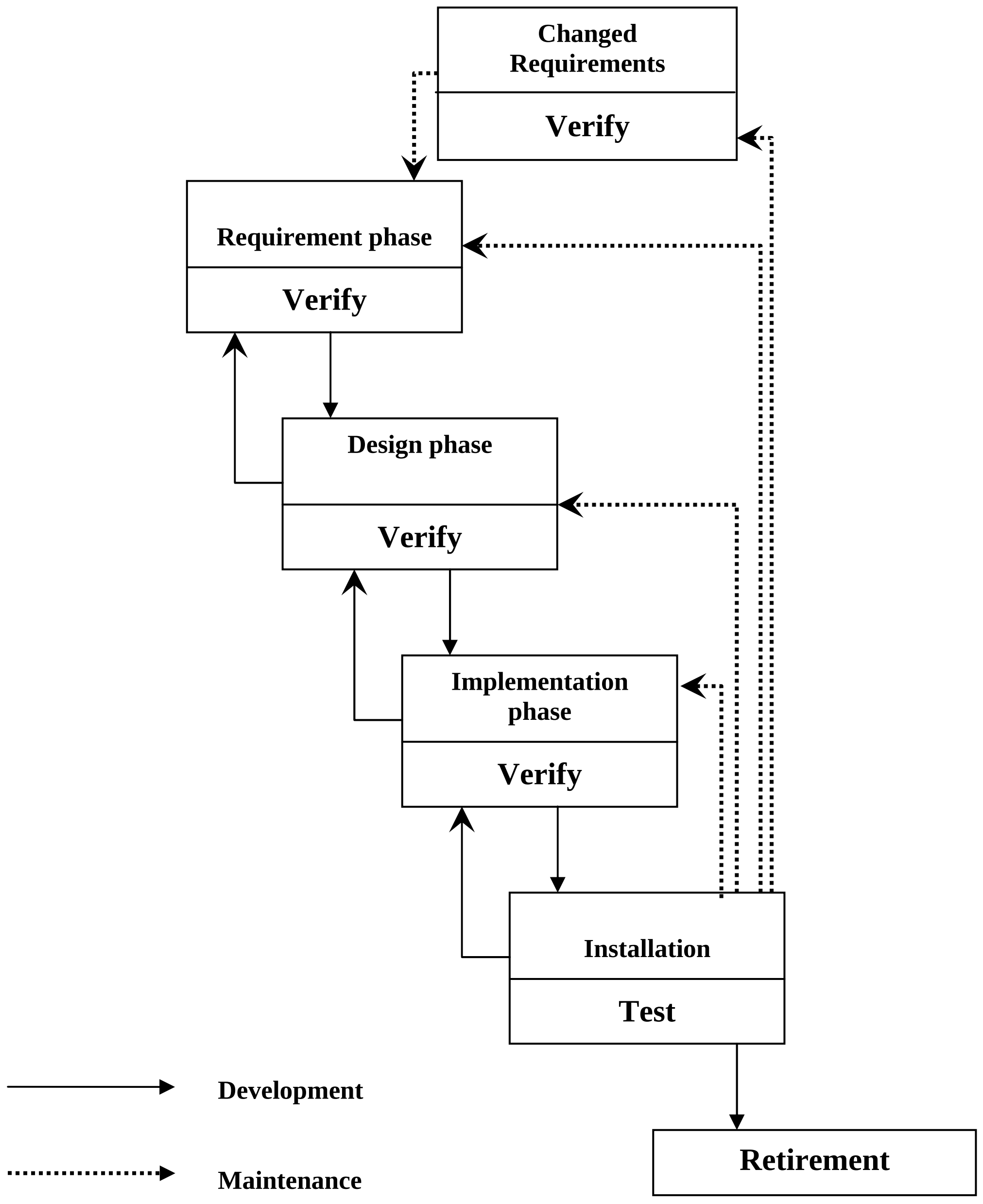
Between these, the first method could have been discarded because of the simple reason that English language does not have any well defined grammar.

This makes the second method the only plausible solution. Due to time constraint the intelligent *language interpreting system* was not designed as a part of the software.

- In the software only plain text files and MS-word files are allowed as an attachment in compressed or uncompressed form. All other file types such as binary files or JPEG files are not allowed as attachments. But if the JPEG or other prohibited file types are inserted into the MS-word file, the system will not be able to recognize and discard them.

4.7 Development standard

The explicit model of the software development process that was followed is waterfall model.
.



The principal ***stages*** of the model are

- **Requirement analysis and definition** : In this stage the system's services, constrains and goals are established by consultation with system user.

- **System and software design**: The design process partitions requirement to either hardware or software systems. It establishes an overall system architecture. Software design involves representing the software system functions in a form that may be transformed into one or more executable programs.

- **Implementation and unit testing** : Unit testing involves verifying that each unit meets its specification.

- **Integration and system testing** : The individual program units or programs are integrated an tested as a complete system to ensure that the software requirements have been met.

- **Installation and maintenance**: Maintenance involves correcting errors which were not discovered in the earlier stages of the life cycle; improving the implementation of system units and enhancing the system's services as a new requirement are discovered.

In each of these stages verification is done as shown in the diagram and in case some flaw is found it is rectified by going back to the previous stages recursively till the source of the problem is identified.
As an example, while the development of the project in the testing phase it was found that the software fails to identify the invalid attachment file-type if the file's extension is changed to some valid file-type (i.e. if invalid attachment file x.jpg renamed as x.txt). To identify the source of the problem we went back to the previous stages of development and it was found that the problem was in design phase where the file-type was being determined by extension of the filename and not by the content of the file. It was redesigned to identify the file-type by the header of the file and the software was implemented accordingly and tested successfully.

5. DETAILED DESIGN

5.1 Design Description

The objective of the project is to design a filter that will parse the content of the mail that is going out of the National Aerospace Laboratories mail server. The filter is designed to be incorporated into the mail system, such that it has access to each and every mail that passes out of the system.

The main goal of the filter is string manipulation and searching for certain strings in a file. The filter has to parse for certain strings and different combination of sub-strings. Also, regular expressions were also to be considered. A regular expression is a pattern that describes a set of strings. Regular expressions are constructed analogously to arithmetic expressions, by using various operators to combine smaller expressions.

Initially, a number of options were considered to achieve this string parsing. After much deliberation, we decided to use the basic filters that come with the Linux environment like Grep, Egrep, Fgrep, Sed, Awk, etc. These basic filters use the Boyer-Moore algorithm, which is one of the most efficient string-matching algorithms available. These basic filters are highly efficient and take minimal time for execution.

5.1.1 Grep

Grep searches the named input files for lines containing a match to the given pattern. By default, grep prints the matching lines. Grep understands two different versions of regular expression syntax: basic and extended. In GNU grep, there is no difference in available functionality using either syntax. In other implementations, basic regular expressions are less powerful.

Normally, exit status of Grep is 0 if selected lines are found and 1 otherwise. But the exit status is 2 if an error occurred, unless the -q or –quiet or --silent option is used and a selected line is found.

5.1.2 Sed

Sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream (a file or input from a pipeline). While in some ways similar to an editor which permits scripted edits (such as ed), sed works by making only one pass over the input(s), and is consequently more efficient. But it is sed’s ability to filter text in a pipeline, which particularly distinguishes it from other types of editors.

Sed commands can be given with no addresses, in which case the command will be executed for all input lines; with one address, in which case the command will only be executed for input lines which match that address; or with two addresses, in which case the command will be executed for all input lines which match the inclusive range of lines starting from the first address and continuing to the second address.

5.1.3 Awk

Gawk is the GNU Project's implementation of the AWK programming language. It conforms to the definition of the language in the POSIX 1003.2 Command Language And Utilities Standard. This version in turn is based on the description in The AWK Programming Language, by Aho, Kernighan, and Weinberger, with the additional features found in the System V Release 4 version of UNIX awk. Gawk also provides more recent Bell Laboratories awk extensions, and a number of GNU-specific extensions.

The command line consists of options to gawk itself, the AWK program text (if not supplied via the -f or --file options), and values to be made available in the ARGC and ARGV pre-defined AWK variables.

5.2 Module Decomposition

The implementation of the filter can be decomposed into the following functional segments.

1. The headers of the mail are processed. All the headers that need to be accessed and processed are done so in the mlfi_header function.
2. The body of the mail is processed.
3. The actual filtering of the mail takes place in the mlfi_eom function, where every section of the mail is parsed for sensitive words.

5.2.1 Processing the headers

The mlfi_header callback function allows access to the header fields. The mlfi_header function is called for every header that is encountered. At this point of time, any specific header can be accessed and altered, if the need arises. The following functions are dealt with in the header module.

1. The name of the user who is sending the particular mail under consideration is identified. Along with this, the destination address is also obtained.
2. If the header is "Content-Type", then the type of subsections in the mail can be identified. We can figure out from the type of Content Type whether an attachment is sent along with the mail. If so, a global flag is set.
3. If the existence of attachment is detected, then the boundary that is used in the mail to separate the different sections of the mail can be extracted. This boundary will be present between the body of the mail and each of the attachments, which helps identify the same.
4. The Subject of the mail can be accessed. This is required to parse the subject for sensitive words at a later stage.

5.2.2 Processing the body of the mail

In the mlfi_body callback function, the body of the mail can be accessed. It has to be kept in mind that when attachments are sent along with the mail, the attachments are encoded in any one of the standards accepted worldwide and appended to the body of the mail. This implies that when the body of the mail is accessed in this function, the body of the mail as well as the attachments in the encoded form will be present. The boundary, that is earlier obtained, will be present between each of these sub-sections of the mail.

5.2.3 The Filter module

The filter that has to run on the mail body has a number of functionalities. These have to be considered in the order of the flow of the data. The filter is called in the mlfi_eom (end-of-message) callback function.

5.2.3.1 The user file creation

The administrator has to maintain a file that contains all the words that are considered to be sensitive for the organization. The administrator also has to group the employees of the organization into groups, according to the hierarchy in the organization. The administrator will have to maintain a list of words for each of these groups. This list contains all those words that a person in the group has authority to use.
When the user’s name is identified, all the groups that he belongs to can be obtained. This gives us a list of words that the employee has the authority to use. From these two lists, we can easily derive the lists of words the employee is not allowed to use. This is maintained in a file.

5.2.3.2 Parsing the subject

From the header section, the Subject of the mail can be extracted. This is now compared with the employee’s not-allowed list of keywords. If the subject is found to contain any such word, then the mail is immediately blocked. Else, the processing of the mail continues.

5.2.3.3 Parsing the body

We have the body of the mail in a temporary file. This body is now scanned for words using the employee’s not-allowed list of words. As in the previous case, the mail is sent through if the body of the mail does not contain any sensitive word. In the case that such a word is used by the employee, the mail is blocked.

5.2.3.4 Derive the attachment files

We have earlier on identified the presence or absence of attachments. If attachments are not present, the filtering process is completed. Else the attachments have to be separated from the body of the mail.
Since the attachments are by default encoded before they are sent out on the network, they have to be decoded before the attachment content can be parsed. This is done by calling a tool called ripMIME that decodes the attachment and separates them into different files.

5.2.3.5 Checking the attachment type

Once all the attachment files are obtained, we consider each of them separately. The type of the file is identified. The administrator has to maintain a file of all those attachments that may pass through the filter. This is referred to identify those attachment types that are considered to be malicious. If such a file is attached, the mail is blocked.

5.2.3.6 Classify the attachments

The attachments that are allowed are of different types. We group all those attachments that are of a particular kind. We then process a set of attachments at a time.

5.2.3.7 Process the attachments

We have identified groups of attachment file types. We now consider each of them separately.

5.2.3.8 Process a text file

The text file can be opened and parsed without much difficulty. The body of the file is scanned for words that are present in the user’s not-allowed list. If the body of the file is found contain any such word, the mail is blocked.

5.2.3.9 Process a Microsoft Word (.doc) file

Since we cannot open a .doc file, we have to use a tool to do the same. The tool we use, Antiword, helps open a word document and display it in a text format. Once a textual format is formed, the body is again parsed foe words, as in the previous cases.

5.2.3.10 Process a compressed file

The employee may also send a compressed file. He may use different forms of compression to form files of the kind .zip, .tar, .gz, tar.gz, etc. each of these can be uncompressed in different ways.
When a compressed file is uncompressed, we may further get different files or more compressed files. We have to consider recursive uncompressing. As we encounter regular files, we parse the body. This is carried for every file in the set of attachments sent.

5.2.3.11 Block the mail

If, at any stage, the mail is found to contain a sensitive word, then the mail has to blocked. This is done as follows.

The mail is captured and its “Recipient “ address is changed to the “From “ address. This ensures that the mail is sent back to the employee who sent the offensive mail. The subject is changed to an appropriate title.
A warning message is written into the body of the mail. This is then appended with the original body of the mail. A copy of the same mail is sent to the administrator. The employee is alerted that the mail did not go through the filter.
By sending a copy of the mail to the administrator, we ensure that the offense does not go unnoticed. The administrator is now in a position to take the necessary legal action.

6. DEVELOPMENT TOOLS

In order to implement our filter, certain development tools that are available on the internet are used. The necessity of these tools arise when we have to deal with certain standards that are followed the world over.
ripMIME and Antiword have to be installed in the Linux system by the administrator. ripMIME can be obtained from www.xamime.com. Antiword can be obtained from www.antiword.cjb.net. The other tools like Glade, gzip, tar and gunzip come along with REDHAT Linux cds (or with any standard distribution of Linux).

6.1 ripMIME 1.3.0.3

When a mail with an attachment is sent over the network, the attachment is encoded using certain standards that are universally accepted and followed. The attachments are encoded to ensure that all the information that is sent over the network will be in certain standard form, like an 8-bit stream. To guarantee that there is some uniformity in the format of information sent over the network, encoding techniques like Base64 encoding, quoted-printable format, Uuencode, etc are defined in the RFC1024 (Request For Command). This encoding is done by the Mail User Agent.

To deal with a variety of encoding that is conventionally followed, a third-party software ripMIME is used. ripMIME is a small program, which has been developed as part of the commercial Xamime development. ripMIME has been written with one sole purpose in mind, to extract the attached files out of a MIME encoded email package. ripMIME was written by Paul L Daniels of PLDaniels.

When ripMIME is run on the mail that is received by the Mail Transfer Agent from the Mail User Agent, the tool will help segregate the body of the mail and the different attachments that are sent along the mail. The tools hands us the attachments in the form that is downloaded when the mail is received. Hence the attached files are extracted out of the MIME encoded package.

6.2 Antiword 0.33

Microsoft® Word files are a very popular form of document that is sent along with the mail as attached files. The filter that is implemented works on the Linux platform. By default, Linux cannot open a Word document. This is because Word documents are encoded in a particular format.

In order that the filter scans through the attached files, it is necessary to open the Word document. Hence we use third-party software to do this for us. Antiword is an application to display Microsoft® Word files. When Antiword is handed a .doc file, it will convert it into a text format, so that Linux can display and parse the document.

6.3 Glade 2.0.0

Glade is a RAD tool to enable quick & easy development of user interfaces for the GTK+ toolkit and the GNOME desktop environment. It also contains built-in support for generating the C source code needed to recreate the interfaces.

The user interfaces designed in Glade are stored in the well-known XML format, enabling easy integration with external tools. Several tools are already available which can turn the XML files into source code in other languages such as C++, Perl and Python. Glade is primarily aimed at Linux and other UNIX variants.

6.4 Unzip 5.50

Unzip will list, test, or extract files from a ZIP archive, commonly found on MS-DOS systems. The default behavior (with no options) is to extract into the current directory (and subdirectories below it) all files from the specified ZIP archive.

Unzip's default behavior may be modified via options placed in an environment variable. This can be done with any option, but it is probably most useful with the -a, -L, -C, -q, -o, or -n modifiers: make unzip auto-convert text files by default, make it convert filenames from uppercase systems to lowercase, make it match names case-insensitively, make it quieter, or make it always overwrite or never overwrite files as it extracts them.

6.5 Tar

Tar is an archiving program designed to store and extract files from an archive file known as a tar file. A tar file may be made on a tape drive, however, it is also common to write a tar file to a normal file. The first argument to tar must be one of the options: Acdrtux, followed by any optional functions. The final arguments to tar are the names of the files or directories , which should be archived. The use of a directory name always implies that the subdirectories below should be included in the archive.

6.6 Gunzip 1.3.3

Gunzip takes a list of files on its command line and replaces each file whose name ends with .gz, -gz, .z, -z, _z or .Z and which begins with the correct magic number with an uncompressed file without the original extension. Gunzip also recognizes the special extensions .tgz and .taz as shorthands for .tar.gz and .tar.Z respectively.

Files created by zip can be uncompressed by gzip only if they have a single member compressed with the deflation method. This feature is only intended to help conversion of tar.zip files to the tar.gz format. To extract zip files with several members, use unzip instead of gunzip.

7. IMPLEMENTATION

7.1 Configuring filter to work in sync with Sendmail

The MTA i.e. sendmail has to be configured so that before it transfers the mail to its appropriate destination it is scanned by the mail-filter.

- **Configuring sendmail:**
  This can be done by modifying the sendmail's configuration file sendmail.cf which guides the working of sendmail. The sendmail.cf should be rewritten such a way that it calls the mail-filter every time it opens a connection for mail transfer.

- **The mail-filter being used:**
  The mail-filter is designed by sendmail's Content Filter API (Milter) which is used to access mail messages as they are being processed by the Mail Transfer Agent (MTA), allowing them to examine and modify message content and meta-information. Milter (Mail Filter) is both a protocol and a library.

  Integration of the mail-filter with sendmail.cf

  To make the filter work with sendmail we can do either of the following:

  1. Add the following to the sendmail.mc file

  ```
  INPUT_MAIL_FILTER(`/usr/sbin/sfilter', `S=unix:/var/run/f1.sock, F=R')
  ```
  and compile the file with the m4 macro processor to obtain the new modified sendmail.cf file by executing the following command

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

  2. Add the following line of code directly to the sendmail.cf file

  ```
  X/usr/sbin/sfilter1, S=unix:/var/run/f1.sock, F=R
  ```

7.1.2 Communication between sendmail and the mail-filter

After making necessary changes in the sendmail.cf file the sendmail program is restarted. The following command needs to be executed to restart sendmail:

```
# service sendmail restart
```
This ensures that sendmail now runs according to the new configuration file. Now the mail-filter is run using a local socket on the same machine on which sendmail is running as a daemon process.
The following command ensures that the mail-filter communicates with the sendmail daemon process using a local socket:

```
# /usr/sbin/sfilter  –p local:/var/run/f1.sock
```

The sendmail and the mail-filter communicate with each other by using the local socket (f1.sock) passed as a command line parameter to the mail-filter program.

7.1.3 Running the mail-filter as a daemon process during system boot

After making the necessary changes in the sendmail configuration file, we need to ensure that the mail-filter starts running as a daemon process as soon as the sendmail daemon starts running. In order to do this, we need to modify the sendmail startup script (/etc/rc.d/init.d/sendmail).

Modifying /etc/rc.d/init.d/sendmail

The startup script contains two functions: **start()** and **stop()**

Start function:

Inserting the following lines after the sendmail client startup command ensures that the mail-filter (Sfilter) also runs as a daemon process.

```
echo -n $"Starting sfilter: "
        /usr/sbin/sfilter -p local:/var/run/f1.sock > /dev/null 2>&1 &
        RETVAL=$?
echo -n
        [ $RETVAL -eq 0 ] && success "Sfilter startup" \
                failure "Sfilter startup"
echo
```

Stop function:

Inserting the following lines after the sendmail client kill command ensures that the mail-filter is also killed when the sendmail client and sendmail daemon processes are killed.

```
    echo -n $"Shutting down sfilter: "
        killproc sfilter
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ]
```

Once the above specified instructions have been carried out, you can be sure that the mail-filter (Sfilter) run as a daemon process on boot up. Sfilter will now communicate with sendmail using a local socket and it will filter all outgoing mails.

7.1.4 Warning

If the Sfilter process is killed by the superuser or any authorized user, sendmail will no longer accept SMTP connections, since the mail-filter is not available. To ensure the proper working of sendmail, run the Sfilter from command line as mentioned at the beginning of this chapter.

If the user explicitly kills the sendmail daemon, then he should also kill the Sfilter daemon to avoid any discrepancies.

8. TESTING

Testing was carried out iteratively at every stage of development. Modules were tested on the basis of certain module specific test cases that were designed to examine their proper working. A combination of these test cases was used to carry out the *integrated testing* of the modules.

8.1 Unit Testing

Modules	Cases	Input	Expected Output	Output	Test status
Checking the subject	1	"Subject : Hi" (containing no unauthorized keyword)	Mail is delivered	Mail is delivered	Successful
	2	"Subject: Saras specification" (containing unauthorized keyword "Saras")	Mail is not delivered and it is sent back to the user	Mail is not delivered and it is sent back to the user	Successful
	3	"Subject : Tbsbt specification" (simple substitution cipher text of the unauthorized keyword "Saras")	Mail is not delivered and it is sent back to the user	Mail is delivered	Unsuccessful
Checking the body	1	"how are you" (containing no unauthorized keyword	Mail is delivered	Mail is delivered	Successful
	2	"Design of saras" (containing unauthorized keyword "saras")	Mail is not delivered and it is sent back to the user	Mail is not delivered and it is sent back to the user	Successful
	3	"design of Tbsbt " (simple substitution cipher text of the unauthorized keyword "saras")	Mail is not delivered and it is sent back to the user	Mail is delivered	Unsuccessful

Checking attachments	1	Prohibited attachment types such as JPEG file	Mail is not delivered and it is sent back to the user	Mail is not delivered and it is sent back to the user	Successful
	2	A plain text file or MS-word file containing unauthorized keywords	Mail is not delivered and it is sent back to the user	Mail is not delivered and it is sent back to the user	Successful
	3	Prohibited attachment types such as JPEG file inserted on a MS-word file	Mail is not delivered and it is sent back to the user	Mail is delivered	Unsuccessful
	4	Prohibited attachment types such as JPEG file compressed by zip, tar or gzip	Mail is not delivered and it is sent back to the user	Mail is not delivered and it is sent back to the user	Successful
	5	Compressed files(.zip, .tar or .gzip) containing prohibited attachments that are again compressed by Zip, Tar or Gzip (ex. A zip file that contains a .exe file that is again compressed by Zip, Tar, Gzip)	Mail is not delivered and it is sent back to the user	Mail is not delivered and it is sent back to the user	Successful

9. Conclusion

Software that filters mails has been developed, keeping the existing mail system at the National Aerospace Laboratories in mind. This system has been incorporated into the mail system that was previously functioning at the National Aerospace Laboratories. With this, every email that goes out of the organization will go through the filter designed by us.

This project helped us gain deep insight into the working of a mail server and the mail delivery system on the whole. It provided an opportunity to work at the system-level. It also helped us understand how to configure a mail server. Knowledge of the intricate details involved in the working of the Local Area Network was necessary to implement the system.

Working on this project has also given us an insight into the conventions and standards followed by the industry while developing a package. We have tried to implement these software practices within the confines of our working environment. Developing an independent system has helped us delve into the complexities involved in software development and testing.

Indeed, the topping of the pudding was the enthusiasm and teamwork put in by the members. The intellectually charged atmosphere inspired everyone to put on their thinking caps and come out with sharp ideas that were crucial to shaping this software.

10. FUTURE ENHANCEMENTS

The system that is developed is a basic mail filter. There is scope for improvisation.

- The filter works on words and phrases. This may be enhanced to take the grammar of the language and the context of the information exchange into consideration. This would result in developing an intelligent language interpretation system that has a wider scope of application.

- Only the basic file types have been handled. The system may further consider file types like .rtf, .html, .pdf and various media formats. The system may also be enhanced to handle graphical inputs in Word documents.

- The employee may choose to encrypt his message in many forms. This may be considered to an extent, though developing such a fool-proof system is highly complex and research oriented.

References

1. Sendmail **-**By *Bryan Costales*
Eric Allman
2nd Edition O'Reilly Publications August 1999

2. Official Red Hat Linux System Administration -By R*ed Hat*

Wiley Publishing, Inc. 2003

3. Red Hat Linux 8 Unleashed -By *Bill Ball*
Hoyt Duff

Sams Publishing 2003

4. www.sendmail.org - for documentation on Sendmail v 8.12.x

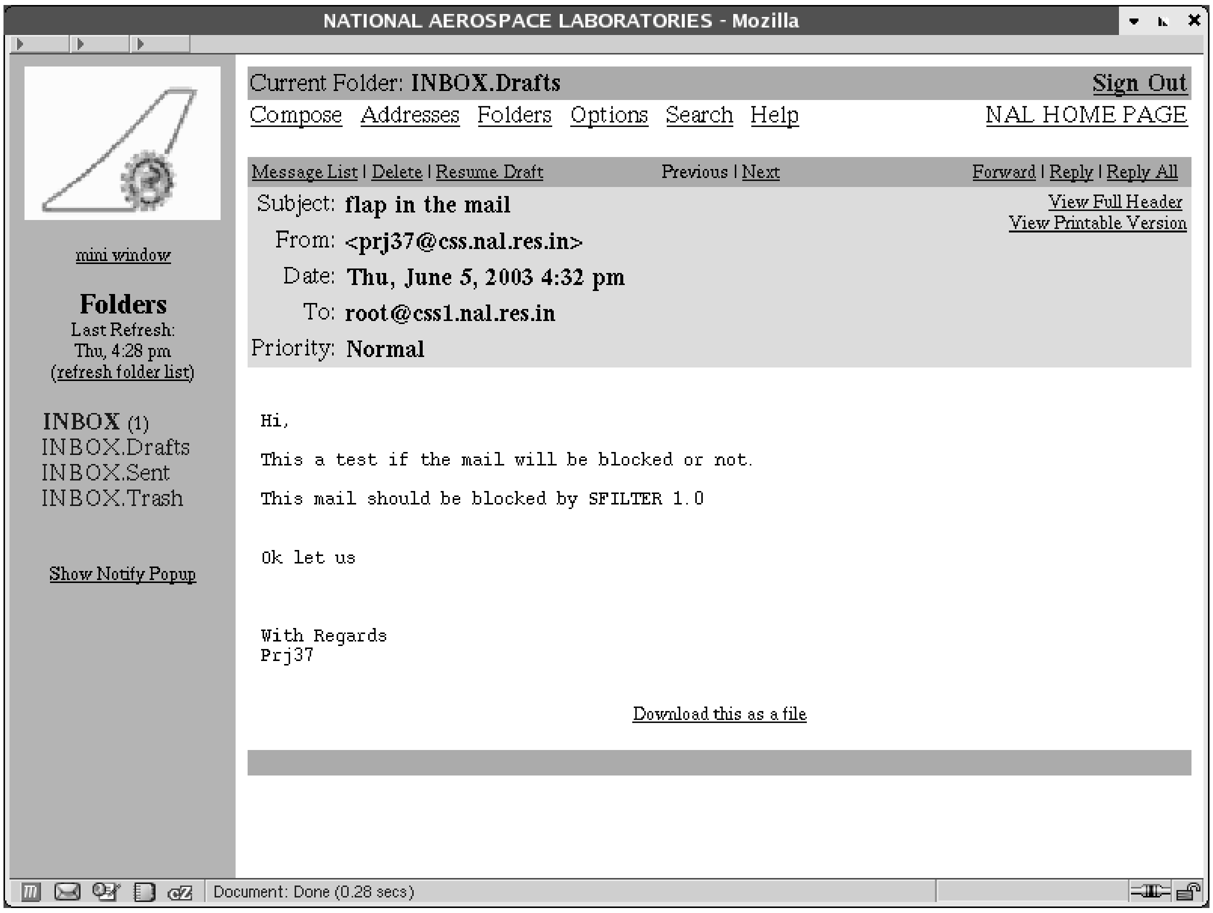
5. www.gnu.org - for documentation on the GNU m4 macro processor

6. www.kplug.org/glade_tutorial - for the documentation on Glade.

7. www.developer.gnome.org/doc/API/2.0/gtk - for documentation on GTK.

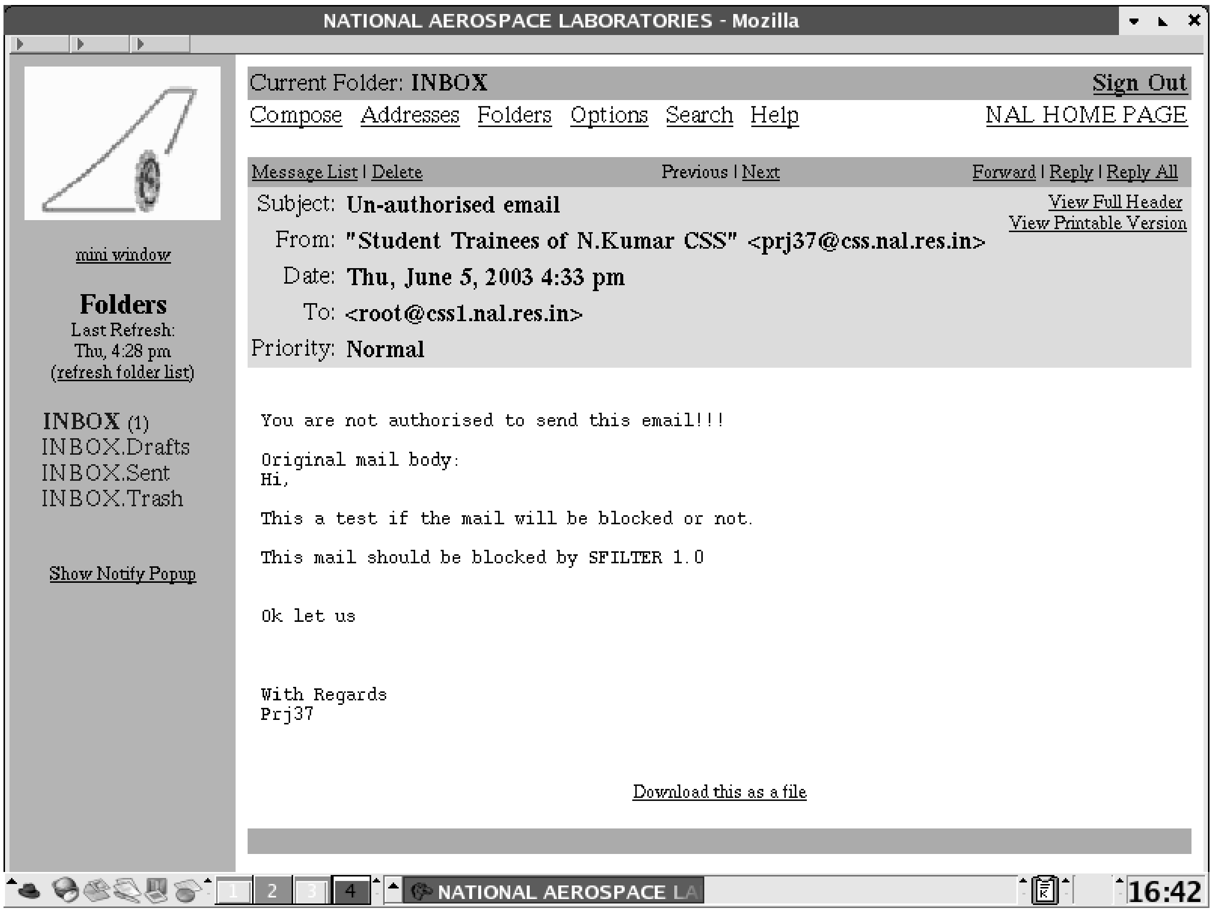
APPENDIX

Appendix A: Sample Screens


NATIONAL AEROSPACE LABORATORIES - Mozilla
mini window
Folders
Last Refresh: Thu, 4:28 pm
(refresh folder list)
INBOX (1)
INBOX.Drafts
INBOX.Sent
INBOX.Trash
Show Notify Popup
Current Folder: INBOX.Drafts
Sign Out
Compose Addresses Folders Options Search Help
NAL HOME PAGE
Message List | Delete | Resume Draft
Previous | Next
Forward | Reply | Reply All
View Full Header
View Printable Version
Download this as a file
Document: Done (0.28 secs)

Subject: flap in the mail
From: <prj37@css.nal.res.in>
Date: Thu, June 5, 2003 4:32 pm
To: root@css1.nal.res.in
Priority: Normal

```
Hi,

This a test if the mail will be blocked or not.

This mail should be blocked by SFILTER 1.0


Ok let us



With Regards
Prj37
```



Subject: Un-authorised email
From: "Student Trainees of N.Kumar CSS" <prj37@css.nal.res.in>
Date: Thu, June 5, 2003 4:33 pm
To: <root@css1.nal.res.in>
Priority: Normal

```
You are not authorised to send this email!!!

Original mail body:
Hi,

This a test if the mail will be blocked or not.

This mail should be blocked by SFILTER 1.0


Ok let us



With Regards
Prj37
```

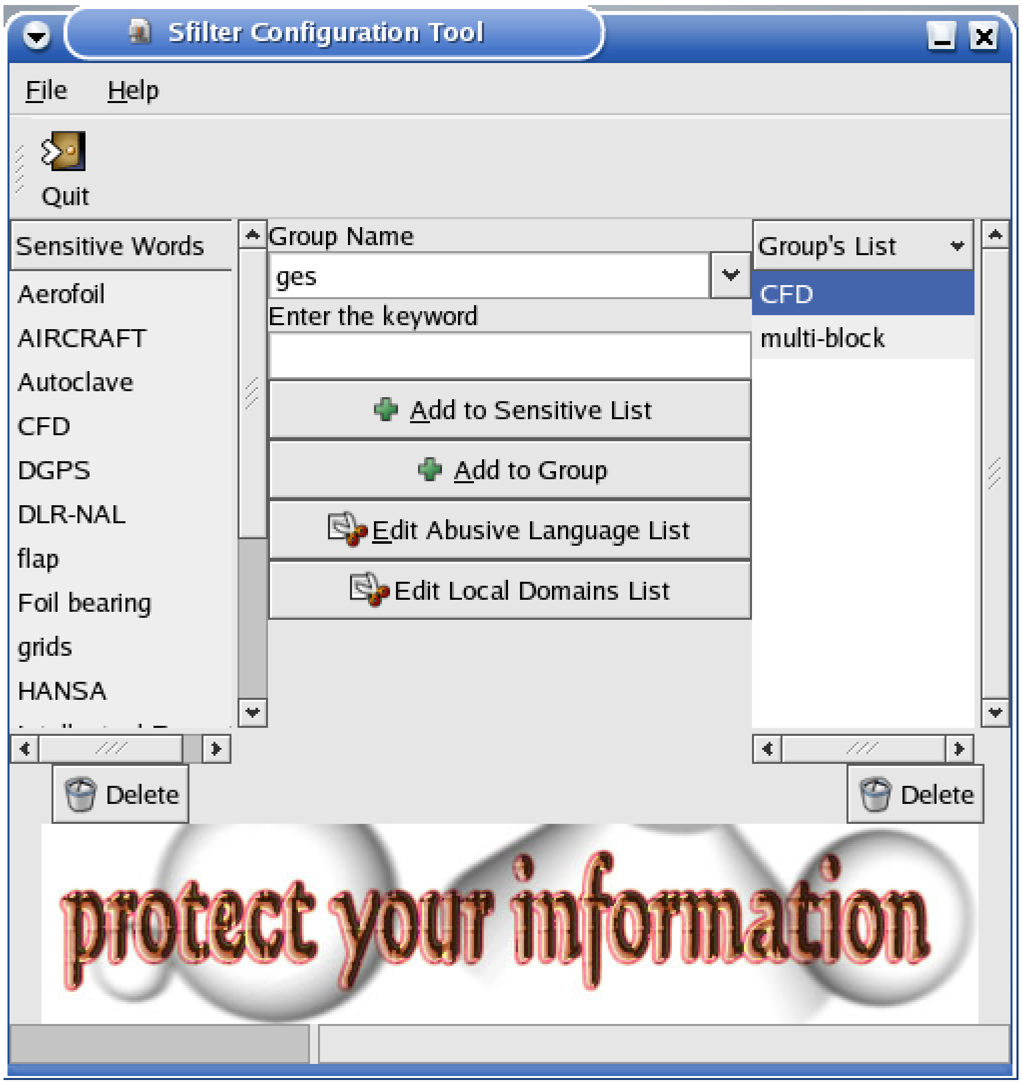


Figure: Sfilter Configuration Tool (Main Window)

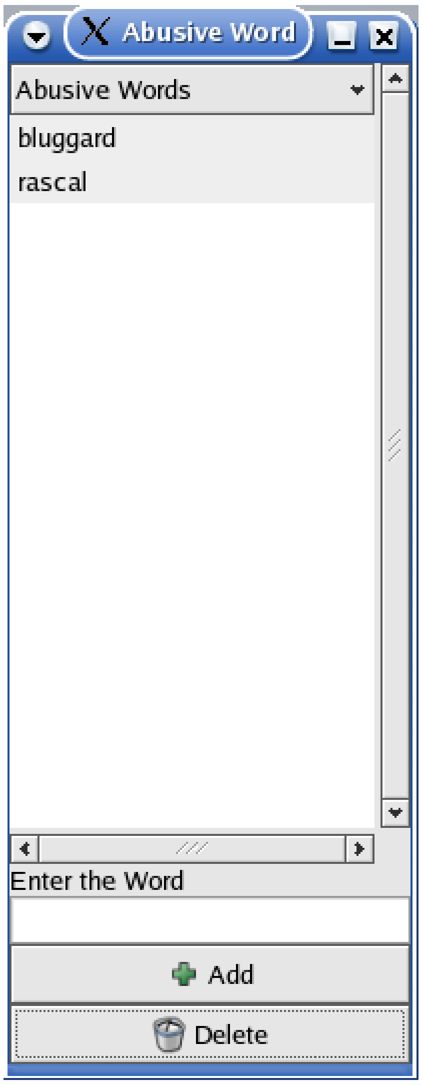


Figure: Sfilter Configuration Tool (Abusive Language List)

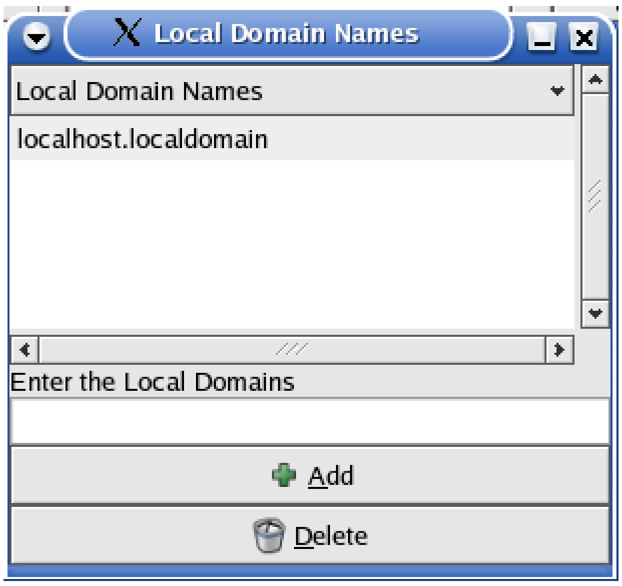


Figure: Sfilter Configuration Tool (Domain Names List)

Appendix B: Installation Guide

RipMIME

To install RipMIME, the following steps have to be taken.
As root, run

- Make a suitable directory like '$HOME/src/ripmime' and copy the 'ripmime.tar.gz' file to this directory.
- Decompress: 'gunzip ripmime.tar.gz'
- Unpack: 'tar xvf ripmime.tar'
- make
- make install

  In the ripMIME folder. Make install will install the ripmime binary to /usr/local/bin.

Antiword

To install Antiword, the following steps are to be taken.

- Make a suitable directory like '$HOME/src/antiword' and copy the 'antiword.tar.gz' file to this directory.
- Decompress: 'gunzip antiword.tar.gz'
- Unpack: 'tar xvf antiword.tar'
- Compile: 'make all'
- Install: 'make install'. This will install Antiword in the $HOME/bin directory.
- Copy the file called 'fontnames' and one or more mapping files from the Resources directory to the $HOME/.antiword directory

Appendix C: Definitions, Acronyms, and Abbreviations

- MUA – Mail User Agent is any program that users run to read, reply to, compose and dispose of email.
- MTA – Mail Transfer Agent is a highly specialized program that delivers mail and transports it between machines.
- MDA – Mail Delivery Agent is a program that does the actual delivery of mail.
- Sendmail – Sendmail is the most popular MTA in the Linux platform.
- SMTP – Simple Mail Transfer Protocol. The language used by MTAs to talk to each other.
- Milter APIs - Mail Filter API (Milter) provides an interface for third-party software to validate and modify messages as they pass through the mail transport system.
- MIME – Multipurpose Internet Mail Extensions describes how messages are sent on the Internet.
- m4 macro processor – The m4 macro processor converts the sendmail.mc file into the corresponding sendmail.cf file.
- Glade - Glade is a tool to enable quick & easy development of user interfaces.
- GIMP – This is the Graphical image manipulation tool used.
- GTK+ – GIMP toolkit used.
- GDK – Graphical drawing kit.

Appendix D: Control File Templates

The software Sfilter (sendmail mail filter) uses *six* types of control files.

- **Universal list of sensitive words (sensitive_list.scf)**

  This file contains a list of all the keywords that are considered *sensitive* for an organization. It is modified and maintained by the system administrator as per the organizational policy through the Sfilter Configuration Tool (A GUI based GNOME tool Sfilter). A sample entry in this file is given below:

  Flap
  Autoclave
  SARAS
  Intellectual Property
  PATENT
  .
  ..
  …

- **List of abusive words (foul_language.scf)**

  This file consists of all the words that are considered abusive and offensive as per organizational policy. Modification and maintenance of this file is done using the Sfilter Configuration tool. A sample entry in this file is given below:

  Idiot
  Bluggard
  .
  ..
  …

- **List of sensitive words for each group (group_name.scf)**

  This file contains a list of all the keywords that are identified as *sensitive* for a particular group of users. This is also modified and maintained by the System Administrator using the Sfilter Configuration Tool. A sample entry in this file is given below:

  PATENT
  Intellectual Property
  .
  .
  …

- **Dynamically generated list of unauthorized words for a group (group_na.scf)**
  This file is generated during the process of email filtering by comparing sensitive_list.scf and the corresponding group_name.scf of the user (belonging

to a group). The entry in this file will be all the words that are unique to the file sensitive_list.scf (all the words in sensitive_list.scf and not in group_name.scf). A sample entry in this file is given below:

```
Flap
Autoclave
SARAS
.
.
…
```

- **List of all allowed attachment file extensions (regex.scf)**

  The file contains the list of all the file extensions that Sfilter can handle. This file cannot be modified by anyone (not even the system administrator). The entries follow the MIME format of specifying file types. A sample entry in this file is given below:

```
    text/plain,? ?[a-zA-Z]*; charset=us-ascii
application/msword
.
.
…
```

- **/etc/group**

  This is a system file that is used by Sfilter to extract all the user groups that exist on the system. Entries are added into this file when new users and groups are created by the *useradd* shell command. This file is used by the **S**filter **C**onfiguration **T**ool to identify all the user groups that exist on the system. The file is also used to identify the group of the particular user that is sending the email, in order to obtain the corresponding *group_name.scf* file. Entries in this file are as shown below:

```
root:x:0:root
prj37:x:500:ges,sagnik,ramya,Debdas
.
.
…
```