Integration of a protection system from undesirable and dangerous content into mail servers

DIPLOMA THESIS PAPER

Department of computer science and business administration
HTBLA Leonding
A – 4020 Leonding, Limesstraße 12-14

Markus Wallerberger
V. DHD / № 26
Linz, May 2004

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Linz, May 2004 Markus Wallerberger

Abstract

This paper deals with the most serious flaws of electronic messaging: undesirable messages, infected content, security and privacy problems. Furthermore, it analyses the present countermeasures against these disadvantages. Finally, it will combine these solutions, add some new techniques and describe the process of integrating them into a mail server.

Firstly, this paper underlines the importance of mailing by analysing its main advantages, speed, machine-readability and asynchronity. In addition, it distinguishes mailing from other communication medias by covering information-theoretical aspects, such as delivery delay and amount of carryable information. Then mail's main protocols, POP3, SMTP and IMF, are described and analysed.

Started a few years ago with some nasty messages that were quickly deleted, undesirable content has become a big threat to the functionality of electronic mailing surprisingly quickly. The Internet community, usually known as very quick, didn't react appropriately. The problem of spam and viruses in e-mails was not fought seriously for years. The advantages for “spammers”, senders of undesirable content, are clear: complete anonymity and no costs for delivery of the commercial messages.

Actually, the present countermeasures consist of filters. This paper examines their functionality as well as their advantages and disadvantages. It issues the battle of filtering techniques versus spamming techniques and the focusses on a complete solution for the problem “spamming” by taking away anonymity and cheap delivery for spammers.

Concerning security and privacy issues, the paper introduces two new protocols: Firstly, XMTP, a SMTP extension, which provides authentication mechanisms and possibilities for back-tracking a message over mail relays. Secondly, a authentication mechanism implemented in JavaScript for secure web authentication. Moreover, this protocol also allows to retrieve mail data without the whole HTML structure. These protocols were implemented within a mail server and its JavaScript client, which architecture is also focussed on in this paper.

Preface

Containing some 14,500 more or less English words forming 64 more or less English pages, this work should give a brief overview of the topic mailing privacy and security. Surprisingly, neither the annoying spam flood nor the amount of infected messages let me choose this topic for this paper. I was facinated by the simplicity of this “anarchistic” communication medium that allows quick and powerful textual communication as well as mail bombs and the unlimited deployment of advertisement and viruses.

For this reason, I limited the practical part to about one half of the entire paper. My intention was to browse through the architecture and then take a look at the most interesting parts of the software components, such as the data base, the authentication mechanisms, the newly developed protocol and some filters. I do not think complete V-Modell conform documents would be appropriate for a diploma thesis paper, though they would fill many pages easily.

Finally, I would like to thank my tutor, DI Erich Mayr, for his support and guidance in this project. Moreover, I would like to thank DI Dr. Franz-Xaver Steinparz for sharing his deep insider knowledge of Linux C/C++ with me.

Markus Wallerberger
Linz, May 2004

Table of Contents

Illustration Index

Index of Tables

1 Introduction

You have mail! When logging into a Linux Server or starting a mail client this message always had something positive, but since about three years it only remembers one of the last 20 minutes' mail filtering session, deleting pretended Microsoft Security Updates directly from security@microsoft.com (wow!), deleting explicit offers for various anatomic improvements, deleting the sixty-ninth well-meant warning of a virus, which will destroy all disks lying close to one's computer and deleting the newsletters one could not remember to have ever subscribed to.

Started a few years ago with some nasty messages that were quickly deleted, undesirable and dangerous content has become a very threat to the functionality of electronic mailing surprisingly quickly. While Internet Service Providers and users are breaking down under the sheer amount of "unsolicited messages", it's addition to pocket money easily got for spammers and cheap advertisement to the companies. They reach millions of addresses worldwide and ISPs and users have to pay the deliverance of the content they then get angry at.

The Internet community, usually known as very quick, didn't react appropriately. The problem of spam and viruses in e-mails was not fought seriously for years. Now, as it is an everybody's problem, the symptoms are fought by using more and more sophisticated mail filter. On the other hand, the mails use more and more tricks to fool this filters. This paper will argue that the only way to fight spam in the long run is to take away the two key points: the low costs for spamming and the anonymity of the causers.

Another problem arose with the higher destribution of electronic mailing and the usage of this medium also by rather unexperienced people: the virus threat. After infecting one PC, most of the viruses have the insatiable desire to send itself to all adresses in the concerned person's address book. This paper will argue that permanently updated virus scanners are not the only way to stop the high mail infection rate, but preventing the user from itself is also an proven option.

As the TIME magazine stated in one of its December issues, "no application is both as annoying and critical as electronic mailing is". When taking a look into newly formed companies, this is perfectly true. Within three decades, e-mails has become one of the mostly used communcation medias. The effort, therefore, to stop the threats for this way of communcation is of great importance in order to secure the usability of global message exchange.

The decentralized, nearly “anarchistic”, character of the Intenet makes it hard to find efficient and reliable countermeasures. This paper will discuss both current strategies and strategies used in the past to avoid the problem of electronic bulk. It will then try to merge these strategies and expand it by some ideas and new ways of avoiding and fighting commercial messages.

Chapter 2 will deal with views on electronic messaging. It will cover general attributes that can be assigned to electronic messaging after giving a brief overview of the history of electronic messaging. It will then position electronic mailing among other communication medias by describing information-theoretical aspects. Finally, it will discuss the most important protocols used for global message exchange today.

Chapter 3 will examine the most important problems concerning global message exchange and its current solutions. The theoretical part will be closed with this chapter. The following chapters discuss the implementation component and its algorithms and structures behind. Finally, chapter 6 will give an conclusion and some views on electronic messaging for the future.

2 Views on electronic messaging

2.1 *General Views on electronic mailing*

2.1.1 Historical and present usage of electronic messaging

Electronic messaging firstly had been a byproduct of the ARPANET, “the first large-scale computer net ever built” [1]. The nationwide network was designed in 1962 by the U. S. Advanced Research Projects Agency (ARPA) to support resource sharing between the research faculties. With ARPANET, the United States were heading for a technological advantage in the cold war by shortening development times. Unfortunately, the designers only created infrastructure for sharing programs and never thought of a possibility to communicate with other researchers textually.

Therefore, no e-mails were sent until 1971. Two years after the net went on-line officially, Ray Tomlinson then sent his first two electronic messages over the ARPANET. This new form of electronic interaction between researchers should quickly become “by far the biggest use of the net”. Actually, the idea wasn't that new: When the first e-mails were driven across the country, the possibility to send electronic messages to users on the same computer had existed for almost a decade. These *intra-computer e-mails* didn't need any network to deliver and were popular among their users, so it was only a matter of time when the possibility to send them to users on other hosts was developed “as more of natural phenomena than ... [a] new technology”. “E-mail caught on fast since 1971,” Ian R. Hardy states, “and it's popularity came to many as a rather unexpected shock”.

Although the following development of comfortable message exchange tools made e-mail easy to use for everybody, the success of this technology didn't root in comfortability but more in the fact that it reshaped the ARPANET community completely. The previously impersonal typed communication via secretaries, which was bound to a strict letter structure, very formal language and useless complimentary closes, was substituted by a quick and personal way to reach another person textually. Even imperfect grammar and social barriers were simply wiped away by this substitution of “Snail Mail” that transmitted messages instantly and personally. Mailing Lists, a possibility to share insights with a wider range of people, were developed a few years after. Still, the reshaped society only consisted of scientific personal – the

ARPANET users - in the 1970s, but the development should quickly take over as the network expands.

As Ray Tomlinson stated 1996 in an Interview with Hardy, "things have not changed much since then" [12]. After the e-mail went international with the connection of the ARPANET to networks in Great Britain and Norway in 1973, it has overtaken the Internet and radically changed the way the whole world's way of communication as it had done it with the ARPANET community.

Today, as more and more users join this "global family", many companies are now fully relying on this previously sophisticated gadgetry developed by some computer freaks in the early 1970s. Electronic Mailing fits perfectly into the world of the late twentieth and the early 21st century and is widely taken nearly as certain as writing a letter – an enormous evolution over some three decades [16].

2.1.2 Overview of the benefits of electronic messaging

Although some of the benefits were already included in the historical views (s. Ch. 1.1.1), the following composition includes main pros, covering technical and security issues as well as social and political aspects. This enumeration doesn't claim to be complete.

Following attributes can be assigned to the message transfer:

- *global*: Electronic traffic is not bound to local logistic networks. It's only limitation is the spacial extent of it's transport medium. Due to the ongoing expansion of the Internet globally, electronic messages can be sent and received nearly all over the world.

- *nearly instant*: If it is delivered, electronically transferred content is delivered nearly instantly. Because the value of an average information is at least indirectly proportional to it's transport time, electronic mailing fulfills the requirement of quick textual communication, a gap traditional mailing was not able to fill.

- *asynchronous*: Although telephone, fax and old chat systems meet the condition of global and instant communication, they don't provide the aspect of asynchronism. Electronic messaging, like traditional mailing, allows communication partners not to be on-line at the same time, but to send and receive information at two different points in time.

- *independent and decentralized*: As an advantage of the ARPANET and the Internet, the benefit of not being bound to any transport company also applies to electronic mailing. Moreover, electronic mailing is bound to the ARPANET network structure and therefore does not need any central delivery center.

- *free of additional charge*: In general, electronic messaging does not require any costs additional to the ISP fee. As the costs for sending a traditional mail are very high, this may be a reason for companies to switch to e-mails.

Concerning social aspects, it can be said that electronic mailing simplified textual communication. To go into further detail, following aspects can be assigned to e-mails:

- *informal*: By using a totally new way of communication, the traditional requirements and form for a letter need not to be followed. The contents of electronic messages should become as "fast" as its medium: Clear and short description of the issues became more important than perfect grammar, style, formal structure and complimentary closes.

- *personal*: As a result of the fact that user name and password is necessary for accessing an electronic mailbox, e-mails are usually opened by its recipients directly.

- *fast*: The instant message delivery was only one aspect of the general fastening of textual communication. The informal style simplified and accellerated the procedure of writing letters and shortened the reading time, the personal aspect made it easier for the communication partners to exchange information by eliminating the "man in the middle", who is opening and deploying the mails.

2.1.3 Overview of the weaknesses of electronic mailing

This chapter will focus on the main weaknesses of electronic communication.

The lion's share of the disadvantages of electronic mailing are security/privacy leaks. Most of the security and privacy holes in the message transfer came out of it's original usage within the ARPANET. To be precise, following attributes can be assigned to the message transfer concerning security/privacy:

- *mainly unencrypted and unsafe*: Because the ARPANET community only consisted of people, who knew each other well, security issues never really arose until the

ARPANET expanded worldwide. This had an effect at the protocol standards SMTP (1982) [17] and POP3 (1984) [18], which offer only very low security measures, although they were already published after the worldwide expansion. The messages are transmitted in plain text, read- and storeable by every router they pass. Content encryption via PGP and other techniques never became really popular.

- *mainly of dubious origin*: The SMTP protocol allows clients to give every e-mail address they wish as sender of a message. Moreover, electronic mailing does not provide any encrypted checksums, so the headers providing information about the way a mail went can be manipulated very easily. Most of the messages, therefore, are of "dubious" origin, but many mail users are not aware of this fact.

These facts result in these three well-known problems, which are discussed in the Chapter 3:

- *Unsolicited commercial messages (UCM)*: Also known as "spam", these messages containing advertisement are deployed in masses to previously collected addresses.

- *Unsolicited bulk messages (UBM)*: Also known as "hoaxes", these messages use Social Engineering techniques to make people forward them to other addresses. Traditionally, they contain warnings of viruses or various petitions.

- *Message Viruses*: Also known as "worms" and "trojan horses", infected messages contain embedded active content that is destructive.

Following social problems arose together with electronic messaging:

- *Missing document status*: Because of the low security level and the easy and hard-to-trace reproduction and manipulation, standard electronic mails do not have the status of documents.

- *deletion of historical letters*: As users delete messages to free up disk space, any discernible historical record begins to evaporate [1]. Although this problem has lost much of it's significance due to the fact that nowadays there is enough disk space for creating comprehensive mail archives, within the short history of mailing many important letters and historical records were already deleted because of lack of space.

2.2 Information-theoretical aspects of electronic mailing

2.2.1 Comparison of communication medias

The following composition of globally used and widely available communication medias will give a general overview as well as a comparison to electronic mailing in order to help positioning electronic mailing among other medias.

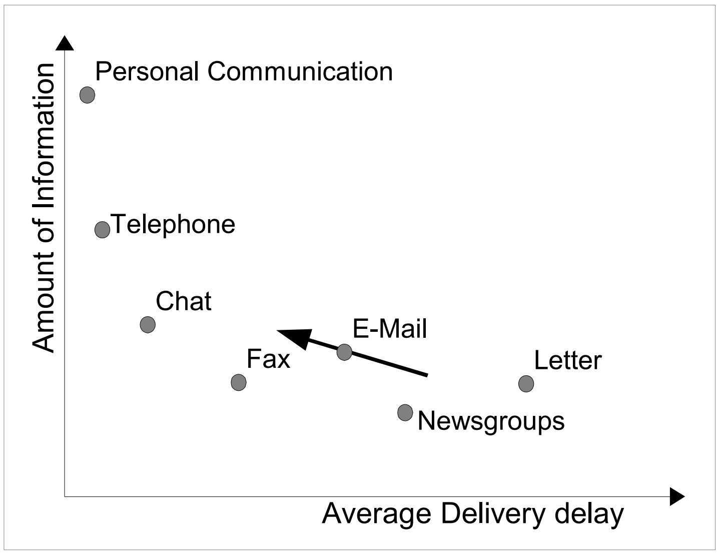


Fig. 1: *Overview over widely available communication medias*

Fig. 1 classifies communication techniques by arranging them within two axis: The *x-axis* represents the *average delivery delay*, meaning the average time difference between the sender's formulation of an information and the understanding of it by the recipient. This time can be splitted into the following time slices:

- *Formulation/Coding time* means the time a sender needs to translate his thoughts to formulations and code this formulation into a message that can be carried by the communication channels.
- *Transmission time* represents the time a message needs to get through one or more communication channels[1] to a point where it can picked up by the recipient.
- *Pick-up time* is an often statistical value, which stands for the expected time a user needs to fetch message from a communication channel. This period also includes the

1 Communication channel in information-theoretical context means "way of transmitting coded information (data)" [19]. Each channel has it's coding type, e. g. voice or plain text, which determines the amount and type of information it can carry. For an introduction to this topic please refer to [20].

interval of message recall. For example, the if the transmisssion time is zero, but the recipient recalls messages only once a day, the average pick up time is 12 hours. (see Appendix A.1 for details).

- *Decoding/Understanding time*: The time the recipient needs for inverting the senders coding actions and extracting the intended information out of the message by decrypting the formulations.

The *y-axis* represents the *amount of information* that is carried through the medium's communication channels. Both axis does not claim to give the exact relation of values between the medias. Especially the x-axis can be seen as slightly logarithmic.

2.2.2 Synchronous communication

Real time communication is not really comparable to the asynchronous principle of electronic messaging. Nevertheless, as many Internet users exchange electronic content in a very low interval, this behavior can be approximated to real time communication (Fig. 1). This way of communicating embraces personal communication as well as telephone, chat and conference systems.

Medias usually provide more communication channels than any other medium. The partners are online concurrently and can exchange information instantly. Telephone or comparable medias provide an additional communication channel, the way of speaking. Moreover, the parties do not have much time to work out the message, which decreases the amount of “blurring” information by using certain formulations. Personal communication offers even more communication channels, like the surrounding, way of appearing, way of acting and so on. These transport by far the hugest amount of information.

In real time communication, transmission and pick-up time are negligible. The coding and decoding times on the other hand are crucial to the correct understanding. Though not very long either, they are the lion's share in this way of transmitting messages.

2.2.3 Asynchronous communication

Delayed communication is the non-native way of transmitting information and is characterized by a longer delivery time than Real time communication and is assigned to electronic mailing as well as fax, newsgroups and traditional mailing.

Asynchronous medias do not provide as many channels as real-time communication does. On the other hand, the asynchronology demands permanent messages, which shrinks the probability of misunderstanding. They can all carry nearly the same amount of information, but can be distinguished by the dilivery time, especially the transmission time and pick-up time.

Due to clear formulations and a restricted message type (textual communication) the time for coding/decoding is negligible. What really matters is the sum of transmission time and pick-up time (Tab. 1).

Medium	*Coding*	*Transfer*	*Pick-up*	*Decoding*
Fax	5 min	0 min	10 min	1 min
Electronic message	5 min	0 min	1 hour	1 min
Newsgroup posting	5 min	10 min[2]	3 hours	1 min
Letter	15 min	36 hours	1 hour	2 min

Tab. 1*: Asynchronous medias with average delivery times*

2.3 Technical views on electronic messaging

2.3.1 Protocol Overview

Two protocols are responsible for the correct message transfer: Firstly, the Internet Message Format (RFC 2822 [22]), which describes the way a syntactically and semantically correct message has to be constructed. The IMF is very complex, but it covers all demands of message transfer (s. Ch. 2.3.3). Secondly, the Simple Mail Transfer Protocol (RFC 2821 [21]), which allows communication partners to exchange messages. The SMTP is very easy to implement server and client applications. One of the problems of SMTP is that it does not implement any security features (s. Ch. 2.3.4).

Two more protocols enable users to check their remote mailbox: The older Post Office Protocol, currently Version 3 issued in RFC 1725 [2], which only allows the most important functions and is, therefore, quite easy to implement. Beside this there is the more powerful, but less popular Internet Message Access Protocol, issued in RFC 1730 [3], which allows advanced mailbox control functions such as multiple folders, partly fetch etc.

2 This is because the most important newsgroups are moderated.

2.3.2 Internet Message Format IMF

The Internet Message Format was designed as a standard for the design and functionality of textual messages of any kind that go over the network. Therefore, it is very flexible and allows much customizing. The IMF is also in use as a basis of the NNTP, which provides the functionality of newsgroups.

An IMF message consists of two major parts: firstly, the headers, which can be splitted into header fields. Each header field offers certain information about the content and consists of a header key, which stores the type of information, and the header value, which stores the information itself. There are some reserved header keys that have a general meaning to all protocols using the internet message format. Moreover, additional header fields can be added by any protocol, client or server. Below, there is an example of an IMF-conform message.

```
Return-Path:        <aon.912692312@aon.at>
X-Flags:            0000
Delivered-To:       GMX delivery to wallerberger@gmx.at
Received:           (qmail 29505 invoked by uid 65534);
   29 Apr 2004 16:13:18 -0000
Received:           from WARSL402PIP6.highway.telekom.at
   (HELO email09.aon.at) (195.3.96.93) by mx0.gmx.net (mx025) with SMTP;
   29 Apr 2004 18:13:18 +0200
Received:           from n944p012.adsl.highway.telekom.at
   (HELO ccckp1xjrd2sv7) ([62.47.61.236])
   (envelope-sender <aon.912692312@aon.at>) by 172.18.5.238
   (qmail-ldap-1.03) with SMTP for <wallerberger@gmx.at>;
   29 Apr 2004 16:13:15 -0000
From:               "5902761000" <aon.912692312@aon.at>
To:                 "'Markus Wallerberger'" <wallerberger@gmx.at>
Subject:            AW: Einladung für Samstag
Date:               Thu, 29 Apr 2004 18:13:20 +0200
MIME-Version:       1.0
Content-Type:       text/plain; charset="iso-8859-1"
Content-Transfer-Encoding:    quoted-printable
X-Mailer:           Microsoft Office Outlook, Build 11.0.5510
Thread-Index:       AcQtWkYfkjNcb7nQSzOHDJte1WF2agAqiu/Q
In-Reply-To:        <opr66msyp6o1s7xm@localhost>
Message-ID:         <20040429161318.29555gmx1@mx025.gmx.net>
```

The second part is called body and carries the real contents of the message. The body may be splitted into multiple body parts or alternatives. The according RFC 2046 [4] permits the following choices and the according **`Content-Type`** Header values:

- **`multipart/mixed`**: The "mixed" subtype of "multipart" is intended for use when the body parts are independent and need to be bundled in a particular order. This is used for attachments.
- The "**`multipart/alternative`**" type is syntactically identical to "multipart/mixed", but the semantics are different. In particular, each of the body parts is an "alternative" version of the same information. This is used for giving multiple formats for different clients, e.g. giving a plain text and a HTML version of a mail.
- **`multipart/digest and multipart/parallel`**: Not commonly used. The provide special forms of multipart/alternative and mutipart/mixed.

2.3.3 Simple Mail Transfer Protocol SMTP

This protocol is the common protocol for exchanging messages over any network that provides electronic mailing. The protocol is relatively simple and easy to understand and to implement.

A SMTP connection works as following: Firstly, a client connects to a SMTP server he wants to exchange messages with. This is done with the commands **`HELO`** and **`EHLO`**. The client may then specifiy sender (**`MAIL FROM`** command) and recipient (**`RCPT TO`** command) and the sends the mail data (**`DATA`** multi-line command). The client may also send a **`TURN`** command to force a server to exchange roles and send the messages he has for the client[3]. Finally, the **`QUIT`** command exits the communcation. The server may respond to each command with FTP-like three-digit status codes.

An example for a SMTP communication is described in Appendix D.1 of RFC 2821.

3 This command is designed for communications over medias, where the number of connections should be minimized. It is, since RFC 2821, deprecated due to Security considerations.

A SMTP Server must change a message: it must add a **`Received`** header to the content and must deliver it to the correct client.

2.3.4 Post Office Protocol POP

In its early times, mails were stored in local files and users were working on the machine they got mail to. Therefore, there was no use for any protocol, which transfers mails from a server to a local client. Nowadays, however, these protocols are absolutely necessary for comfortable message exchange. Nevertheless, the effects of the little need for such a protocol in the beginning of mailing's career led to the lack of security and functionality within POP3. It's only use is transferring and, optionally, deleting inbox messages from a server.

A POP3 protocol implementation works as following: The client identifies itself by the **`USER`**/**`PASS`** command pair. Optionally, the server can force the client to authenticate via the CRAM-MD5 Algorithm, which will be discussed later (s. Ch. 3.6.1), by using the **`APOP`** command. It then can receive messages by using the **`RECV`** or **`TOP`** command. Usually, it will then delete messages (**`DELE`** command) and quit (**`QUIT`** command).

An interesting fact about POP3 is that the server must perform an *exclusive lock* on the mail box when a client connects. The client, therefore, must not see messages that arrive after it's enter. POP3 server may not change messages.

2.3.5 Internet Message Access Protocol IMAP

The IMAP is newer than the Post Office Protocol. It was designed to match the growing needs for more features than POP offers. Although it is very powerful, it hasn't managed to become the mostly used mail delivery over the Net. POP3 still defends this position with great success.

In RFC 1730, four states are defined for IMAP:

- When a client connects, the protocol is in *non-authenticated* state. The client must then authenticate to the server. IMAP allows the server to provide any authentication it wishes by offering the **`AUTHENTICATE`** command or default login with user name/password combination using **`LOGIN`**.

- After authenticating the client, the protocol enters the *authenticated state*. In this state, the client is allowed to select, create, rename or delete mailboxes by using **`SELECT`**, **`CREATE`**, **`RENAME`** and **`DELETE`** command. It can also search newsgroups and may subscribe or unsubcribe to certain newsgroups. When subscribing to a newsgroup, it is simply added to the list of mailboxes. In general, mailboxes are comparable to folders containing messages.
- At selection of a mailboxes, the protocol switches over to *selected state*. Additionally to the commands valid in authenticated state, the client may perform several actions with mails. It may append (upload) an new mail to a mailbox and it may copy or delete it. IMAP offers a wide array of *fetching options* (**`FETCH`** command), for details please refer to RFC 1730.
- When sending the **`LOGOUT`** command, server and client enters *logout state*. The server now performs every action that has had to be delayed to the end of the session and both communcation partners close the connection.

Although it offers a huge amount of functionality, IMAP partly misses a clear and concise protocol structure. The request/response pairs are often confusing and hard to interpret. That may contribute to the low popularity amongst developers and users.

3 Problems of Global Message Exchange

3.1 Views on Unsolicited Commercial Message Issues

3.1.1 Definition

The expression UCM or 'Unsolicited Commercial Message' refers to messages that were created to be deployed in masses to usually millions of electronic message addresses. These messages, commonly known as "spam"[4], therefore, comply with the requirements of advertisement. There are various reasons for causers, which may also be combined, to deploy this junk:

- *"fun"*: Some spammers just want to see how many users they can reach with their messages. This habit is quite similar to hacking (breaking into a system just to see if one can manage it). The processing it's decreasing though.
- *destructive issues*: Messages carrying destructive content are also assigned to the expression UCM, because often they advertise the attachments they carry to the user. They are extremly dangerous, because they often use social engeneering[5] techniques to make the users believe in them.
- *advertisement*: These messages are the great majority. They are advertising for special kinds of products, which you don't get so easily and are often not popular.

3.1.2 Deployment of UCMs

The British application service provider MessageLabs, which committed itself to fighting the problem of unsolicited messages, describes the development of the problem in its June 2003 white paper as following:

> *It began as little more than a "nuisance" around a decade ago, initially frowned upon as being somewhat unethical but nobody felt the need to do anything about it. Since then spam has grown exponentially to become a serious threat to email security for*

4 Spam® is a kind of meat and an popular addition to breakfast in the U.S. Experts believe that it was assigned to unsolicited messages because you get tired of this meat sometime and then it's annyoing you.

5 Social Engeneering in this case means the manipulation of a persons actions and thoughts by faking the social environment, especially identities and surroundings. For more information, please refer to the excellent documentation by Kevin Mitnick, who was in prison for several years for committing crimes connected with Social Engeneering [15].

businesses globally. [...] It's bad, and it's getting worse. That is the clear message. Researchers at Gartner have predicted that spam will account for around 50 per cent of all email by 2004 ... [13]

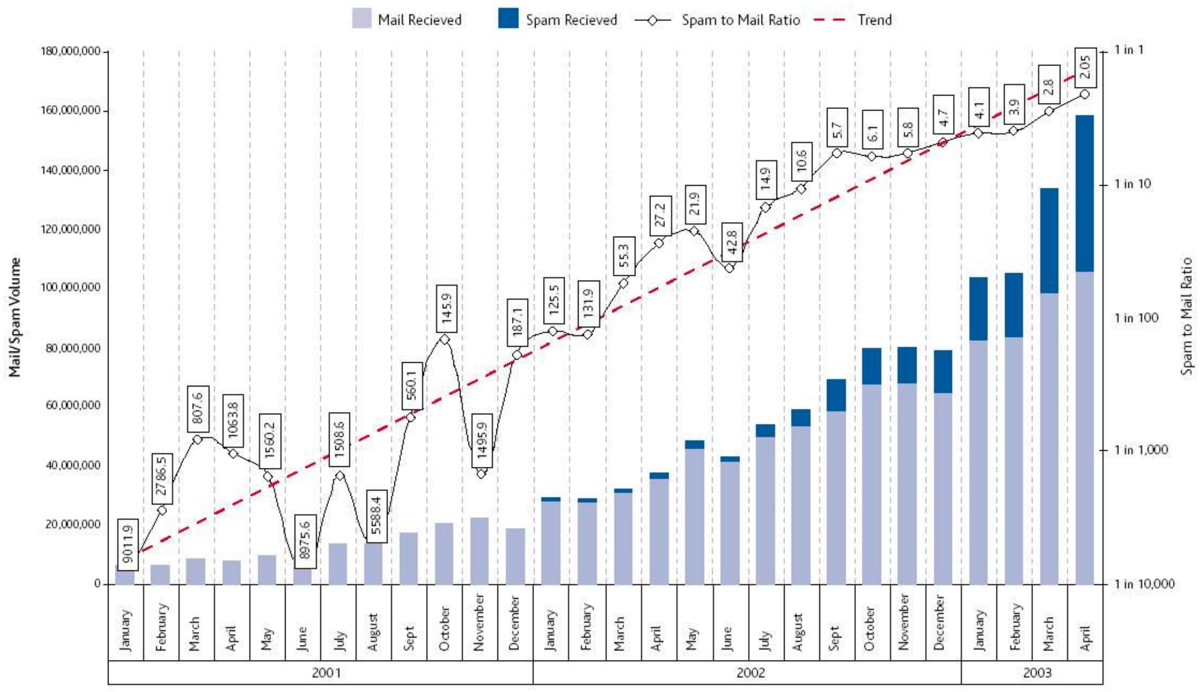


***Fig. 2**: Amount of unsolicited messages caught by MessageLabs [5]*

The Gartner Research Centre should be more than right. Recent measurements by the MessageLabs Intelligence amount the spam to a total of 1 unsolicited message in 1.4 to 1.5 messages, meaning a spam ratio of two thirds. Fig. 2 shows the development of this threat. It should be mentioned that the scale is logarithmical, otherwise the immense increase from 1 to 90,119 to 1 in 2.05 messages within 28 month would not have been possible to be displayed.

3.1.3 Reasons for UCMs

Unsolicited commercial messages provide certain advantages over traditional mailing ads due to the attributes of electronic mailing (s. also Ch. 1.1.1). These attributes are now examined under aspects of UCMs. In general, the reasons can be splitted into two categories:

- *Deployment*
- *Network weaknesses*

These problems will now be issued in more detail:

- *Deployment*: Most of the advantages for spammers nested in the principle of electronic mailing can be assigned to the deployment and way of transmission of the electronic messages.
  - *Cheap deployment*: While at traditional mailing, the price for transmitting the message is paid by the sender (porto), this only partly applies to electronic mailing. The sender usually only pays for the connection, whereas the internet service provider, backbone holder and the mail servers have to pay the costs of delivering the message (Fig. 3).

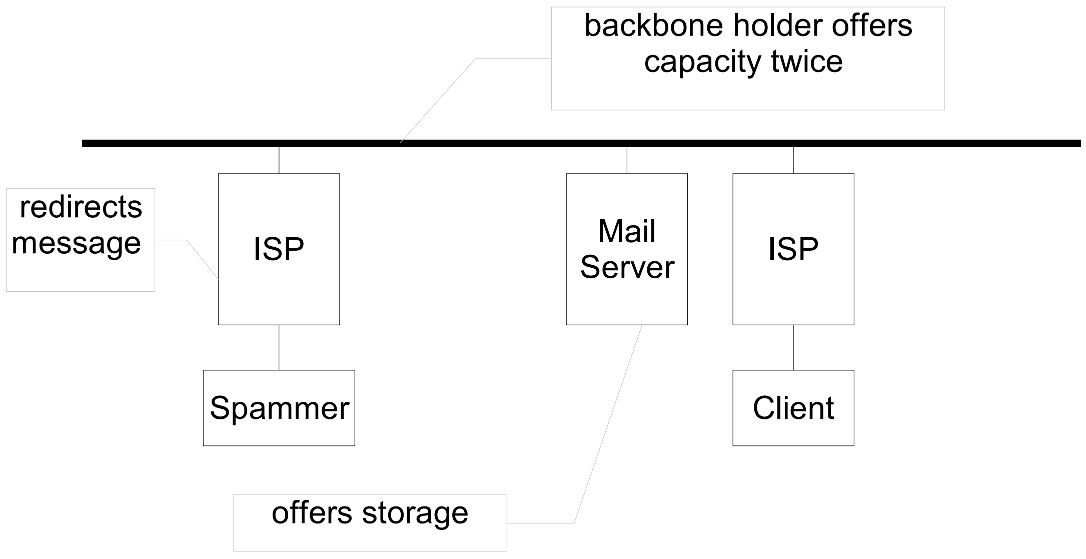


Fig. 3: Cost deployment of UCM transmission process

  - *Remote storage*: Unlike any other providing of online contents, UCMs use the client mail server for storing their advertisment and they, therefore, let the mail server and in case of a paid accound the client pay for unsolicited messages they receive. For this reason, mail servers should also be interested in avoiding and blocking spam in order to reduce the disk space required per user.
  - *Easy deployment*: Unlike traditional mailing, electronic mailing does not need any formatatting, printing or enveloping of the messages. These advantages allow any user without any infrastructure, but only equipped with a mass mailing client, to do any advertising capaign he desires to do. The fact that electronic mails does not need to printed etc. also reduces the costs of messaging. Just to give an example: Let us assume that an advertisement page (usually in colour)

has printing costs of 7 ct. a page, which is quite low. Costs for sending a package under 20 grammes within Austia is 51 ct. So the price for sending a message with 4 pages is 79 ct. an example!

  - *Quick deployment*: As mentioned above, no time for printing and enveloping the messages is needed. This time usually has to be paid for too, which is not included in the example above. Moreover, as mails are transmitted instantly and normally designed rather easily, they can quickly reply to any recent event.

- *Network weaknesses*: The second great reason cluster for unsolicited message issues are network weaknesses connected to the anonymity of the users, which is a result of the decentralized structure of the Internet. As the ARPANET was designed in the 1960s, no one thought of building up authentication mechanisms as every node and its users were well-known.

  - *Network anonymity*: It is very hard to track back an IP address or a domain name to a certain person to make it accountable for its actions. The DNS reverse lookup procedure tries to make this possible, but the problem is that it is not implemented everywhere (s. Ch. 3.4.2 and Ch. 5.2).

  - *Missing SMTP authorization*: SMTP does not provide any measures or protocol elements for a reverse authorizing mechanism. This lack of authorization has two effects: Firstly, a SMTP server authorizing its own users must either trust them or force them to firstly authorize via another protocol. Secondly, a SMTP server must fully trust another SMTP server because it cannot determine if the mail was sent by a valid origin (s. also Ch. 3.6.3).

  - *SMTP relays*: The default configuration of a mail server usually allows SMTP servers to use it as a mail relay, meaning passing on their messages to another server by this relay. By using this technique, spammer can disguise their real identity.

3.1.4 Design of UCMs

The iX magazine [6] gives a very accurate description of the attributes of unsolicited commercial messages assigned to the message and its transfer (Tab. 2).

	"Spam"	***"Ham"***
Origin	*highly diversified*: Spammer often use "throw-away" providers for sending commercial messages.	*nearly equal*: Most of the electronic messages going to one user are sent by only a few origins. This can be used for automatic Whitelisting (s. Ch. 2.2.2)
Contents	*nearly equal*: Commercial messages are often slightly diversified to make it harder for filters to detect these content (s. Ch. 3.6.2). On the whole, however, they contain nearly the same contents.	*highly diversified*: As virtually none of these electronic mails are generated automatically, the contents diverge highly.

Tab. 2: Attributes of Ham and Spam

Some of these messages neither fit Spam nor Ham properties. These messages are called UBMs and will be discussed in Chapter 3.3.

3.1.5 Legal issues

Sending unwanted commercial messages is forbidden under the Austrian law [9]. They fall into the same category as the so-called "Cold Calls", which refer to unwanted telephone calls that are made randomly by Companies.

On the other hand, a mail server, mail relay or mail provider may not delete any message it detects as spam. It must deliver these message as-are to their users or, if they don't deliver them, they must send a message to the recipient that a message was filtered out and offer a link to a download location. This was approved by the German supreme court, where a mail provider was sentenced to pay damages [10].

3.2 Recent countermeasures against UCMs

3.2.1 Overview

Almost every countermeasure against UCMs focusses on the filtering of spam messages because of attributes and properties of the message. They fit quite well, but do not provide a complete protection against commercial messages. The current techniques include:

- *Statistical analysis*: usage of statistical techniques to determine probability for spam
- *Aspect analysis*: analysis of certain attributes that hint on spam.
- *Fuzzy checksums*: method to re-identify once found spam

Therefore, newer techniques rely on sender filtering to eliminate the spammers anonymity. These techniques are also only partly relyable as UCMs senders are switching gateways and mail box names quickly. Some filters, however, use exactly this technique to identify spam. Methods include:

- *Whitelisting*: Array of addresses and gateways, which mail should be delivered from in any case.
- *Blacklisting*: Array of gateways, which are known for spamming
- *ABC method filtering*: Categorizing senders by their mail volume already sent
- *DNS reverse lookup*: Method of verifing the senders domain name.

3.2.2 Whitelisting/Blacklisting

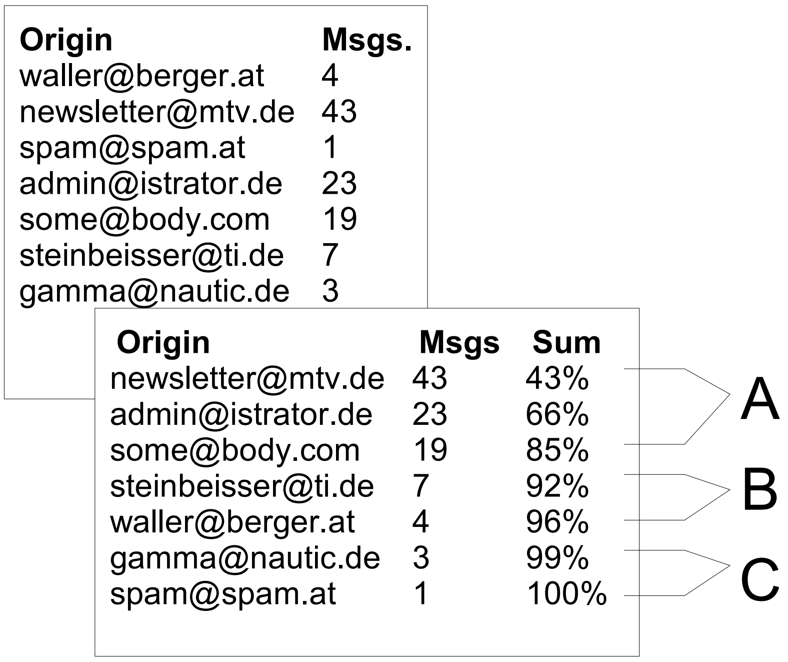
The easiest way to detect spam is the blacklisting/whitelisting technique. IP-addresses and domains, which are known for sending spam are entered into a “Anti-spam list” that is often deployed globally to share this important knowledge. Blacklists are also common on user level, because many users get newsletters or other things so they can ban this addresses.

Whitelisting is mostly done on user level. A user can define domains or addresses that do not send spam. However, also big mail provider as hotmail.com or gmx.at are often entered into a system-wide whitelist because they are known for not sending spam messages. Whitelists can be synchronized with ABC method filtering (below).

3.2.3 ABC method filtering

As the probability for spam is indirectly proportional to the number of messages totally sent by a user to another user, this characteristic can be used for filtering messages that are spam with a high probability.

To do so, the *ABC method* can be used. This algorithm is known from business administration theory to arrange and prioritize certain object because of attributes. Firtly, a numeric attribute must be chosen. The method then classifies the object class A (highest priority), B (medium priority) or class C (lowest priority) [8].



Origin	Msgs.
waller@berger.at	4
newsletter@mtv.de	43
spam@spam.at	1
admin@istrator.de	23
some@body.com	19
steinbeisser@ti.de	7
gamma@nautic.de	3

Origin	Msgs	Sum
newsletter@mtv.de	43	43%
admin@istrator.de	23	66%
some@body.com	19	85%
steinbeisser@ti.de	7	92%
waller@berger.at	4	96%
gamma@nautic.de	3	99%
spam@spam.at	1	100%

Fig. 4: ABC method example

To give an example, we will assume a user has gotten 100 mails totally. Class A range is defined as 80 per cent of total messages, class B range is 95 per cent (Fig. 4). Firstly, the objects are ordered by the attribute values descendingly. After that, the number of messages up to the current row are summed up and divided by the sum of all messages. This number is then classified A, if it started below 80%, B, if it started between 80% and 95%, and C in all other cases. More classes may be inserted if a finer graduation is desired. For example, if all users that have never sent any messages should be distiguished from others, a fourth category “D” could be inserted with a treshold of 100 per cent.

The advantage of this method lies in its simplicity. Disadvantages are that these methods only gives a probability for spam and do not provide a clear decision and that these methods may fail at newsletters that are changing their sender address continously.

3.2.4 DNS Reverse Lookup

Another, non-probability based way, of sender filtering is the DNS Reverse Lookup method. When a DNS lookup is performed, a server verifies that the server the mail comes from holds the address it was sent from. This is done by using the **`in-addr.arpa`** domain. The server simply checks if the IP address the server connects with fits to the domain name specified in the e-mail.

This method is an attempt to eliminate the sender's anonymity. Therefore, it can be logged which domains acted “correctly” and which did not. To get DNS RL working, every domain name server must have its Reverse lookup entry in its zone file.

The advantage of the DNS lookup is that it can identify a sender precisely and needn't rely on the identity given by itself. For that reason, it can watch the user activity and choose dynamically if it is a spammer on not. Disadvantages of the DNS reverse lookup is that mail servers are often used only as a mail relay and do not represent the original domain. Another disadvantage lies in the incomplete implementation of reverse lookup in mail servers. Especially some small DNS servers do not support reverse lookup. Either you treat them as “not valid” or spammers have a security hole to break into your server.

3.2.5 Fuzzy Checksums

Unlike other checksums (e.g. CRC) that provide an exact message digest, a fuzzy checksum is an approximation. For highly redundant content, they, therefore, return the same checksum. As this applies to most of the spam messages, this method can be used for filtering UCMs and may also be used for filtering UBMs, which are described later.

According to the iX magazine, spammers already found another way to escape these fuzzy checksums: they simply add much seemingly accidential content that “poisons” the filter and makes it totally useless.

3.2.6 Statistical analysis

These methods work with Naïve Bayes and other statistical techniques to determine the probability that a message contains advertisement. This probability is calculated by taking a look at certain aspects of a mail that hint on spam.

Spam would be easy to detect if the contents were totally equal. Therefore, senders use advanced techniques to disguise these contents. Sophos Inc. desribed these techniques in their recent whitepaper [7]:

- *Appendices*: Spamming programs include random characters into messages to make it more difficult to detect by spam filters.
- *Black Hole*: Technique of hiding spaces and other masking things by using a very small font. This substitutes the old technique of textual spacing.
- *Numbers game*: Technique of encoding letters by writing them in HTML Entities, e.g. to write not “**`A`**” but “**`&#0041;`**”.
- *Invisible Ink*: Manipulation of font and background colors to make important keywords that point to a nonspam message literally disappear on screen.
- *Slice and dice*: Uses HTML tables to “shred” a message within stripes.

3.3 *Views on Unsolicited Bulk Messages*

3.3.1 Definition

The expression UBMs or Unsolicited Bulk Messages refers to messages, which are intended to be forwarded by users. These messages are often called “Junk” or “Hoaxes”. The lion's share of these messages is sent for fun reasons, only to see where a message once deployed will spread over the globe.

Usual subjects and contents for UBMs are:

- *Virus warnings*: These so-called “Hoaxes” often are said to be from a certain anti-virus software company and contain warning from an extra-destructive virus every user has to know of.

- *Petitions*: These messages often contains short stories or certain facts to recent political topics and call its recipients to take actions for or against these things by putting its name under the mail and forward it.

3.3.2 Deployment of UBMs

There is only little knowledge of the deployment of UBMs as the definition of this expression varies from site to site and the borders for a users what to treat as UBM, and what is UBM, but the user does not know.

Generally, it can be said that as the number of users connected to the Internet and exchanging mail grows, the number of UBMs increases gradually too. On the other hand, in the latest time more and more users know that the messages does not do anything, so they don't forward them.

3.3.3 Design of UBMs

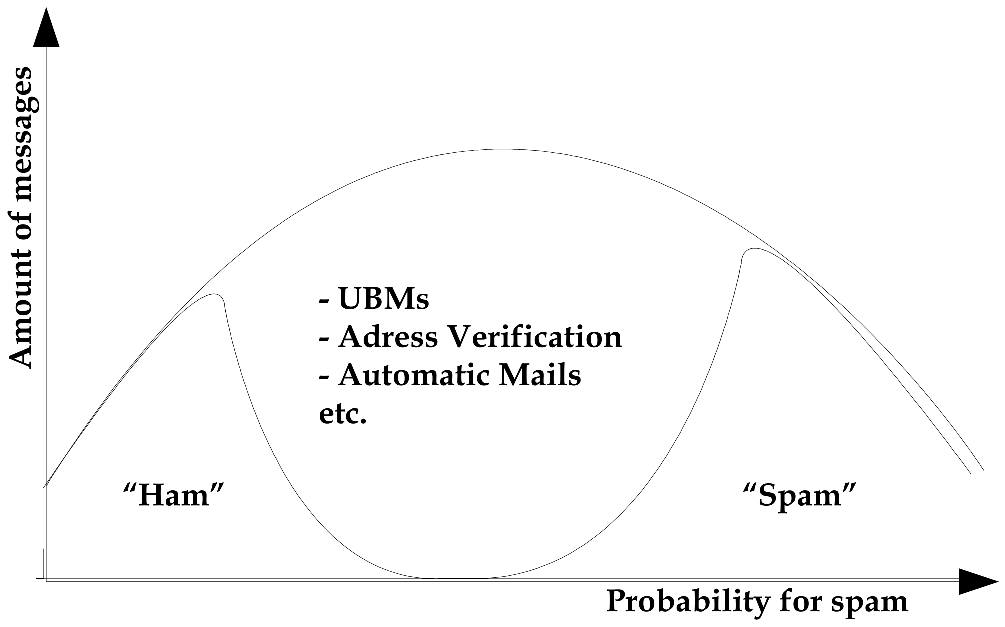


Fig. 5: Problem with filtering UBMs [14]

UBMs with Social Engeneering techniques. The Social Engeneering techniques include:

- *Intendor fake*: The mail says it comes from microsoft.com or un.org or any other organization that is well-known it the sector.
- *Subject fake*: The mail contains false or at least dubious facts that a user should believe in an treat them as important/funny for all users and forward it.

The problem with filtering UBMs is that on the one hand their contents diverge only slightly but they come from known users, so they are between "Ham" and "Spam" in (Tab. 2). This is also illustrated in **Fig. 5**. According to the iX magazine, this share of messages not being ham or spam increases as the amount of automatic messages increases. For this reason, it is very hard to distinguish UBMs and other automatically designed, but necessary mails.

3.4 *Views on infected messages*

3.4.1 Definition

The expression "infected messages" refers to messages, which are carrying dangerous and/or destructive contents (viruses). These contents are intended to be deployed on as many computers as possible. Therefore, in connection with the advantages of electronic messaging (s. Ch. 2.1.2 / 2.1.3) and unsolicited commercial messages (s. Ch. 3.2), e-mail is very good medium to deploy these dangerous contents.

3.4.2 Deployment of infected messages

Infected messages are mainly deployed in two ways:

- *via infected computers' address book*: This way is most commonly used for deploying viral messages. However, it is very dangerous because it is not possible to destinct messages sent by viruses from messages sent by the user. It can even lead sender-based filters ad absurdum and a user opens an infected attachment more easily if it comes from a well-known user.

- *via spam lists*: Recent spam attacks often included some Security updates to avoid more spam or viruses. These attachments often contained viral contents. This method of deployment is also dangerous, because it can effect millions of users at the same time.

3.4.3 Nesting places of infected contents

This chapter will not examine the design of viral contents itself, but will show how messages can nest within a message. Most of these techniques requires HTML mails.

The classic way of deploying dangerous content is via *attachments*. These technique offers an array of advantages for the spammer: It is also available for text-only mails, it doesn't requires easy to track inbound elements and it is hard to detect. Fortunately, users has been alerted and are now opening attachments with care.

Another way of enabling dangerous content is via *Scripting*. This is a very dangerous method against Microsoft Windows users, because Microsoft offers an array of insecure scripting languages. Most of these security holes have been closed though, but by using ordinary JavaScript you can do enough harm by automatically forwarding to another page where some "Security updates" wait or something like that.

Scripts may deployed within the following tags and attributes:

- **`<script> </script>`**
- **`<jscript> </jscript>`**
- **`<vbscript> </vbscript>`**
- Event handlers **`onMouseover, onMouseout, ...`**

The last way of virus deployment is via *inbound active elements*. The least destructive way are inbound images. These images refer to a server that logs if the mail has been opened. The spammer, therefore, can see if the address he sent the message to is valid or not. An example of an image path:

`<img src="http://foo.de/verify/wallerberger.at.gmx.dot.at.gif">`

If the user gets a message, the mail client will automatically query for this GIF image. The appearent directory **`verify`** in this example refers to a cgi script, which may read out the following path and is now knowing that our mail box is valid and open for further attacks of this kind!

Other active elements are more destructive. They include Active X elements, and partly other, less dangerous plugins, like Flash, Shockwave and Java Applets.

3.4.4 Legal issues

In Austria, infiltrating or destroying a computer system is punishable under the Strafgesetzbuch § 118a (Unbefugter Zugriff auf ein Computersystem/Non-allowed access to a computer system) [11].

3.5 Recent countermeasures against infected messages

Virus filters remain the most common way to block infected content. In addition, some mail servers deactivate active HTML content or show HTML version only if the user explicitly wants it.

3.6 Security/Privacy Considerations

3.6.1 POP3 weaknesses

RFC 1725 [2] only gives two weaknesses of POP3 protocol. The first weakness is the plain text transmission of username and password in the **`USER`**/**`PASS`** authentication mechanism. As at least 95% of clients use this authorization, most mail box passwords are buzzing around between the routers, waiting for somebody to read them out. Secondly, no server should return **`-ERR`** (negative return), when a client queries for an invalid user like it is possible in the POP3 specification.

Actually, the *CRAM-MD5 algorithm* described in RFC 1725 for the "secure" APOP-command raises enough security considerations. This algorithm works as following: In its greeting message, the server adds a timestamp under enclosed by sharp brackets "<>". Therefore, the server may respond as following:

```
+OK POP3 server ready <1896.697170952@dbc.mtview.ca.us>
```

The client now uses this timestamp, appends its shared secret and then encrypt the whole string with the MD5 message digest algorithm [13]. In RFC 2617 , where HTTP authentication techniques are described, authors mentioned the much weaker MD5 algorithm compared to the rfc2617 digest authentication.

> *Digest Authentication does not provide a strong authentication mechanism, when compared to public key based mechanisms, for example. However, it is significantly stronger than (e.g.) CRAM-MD5, which has been proposed for use with LDAP, POP*

> *and IMAP (see RFC 2195). It is intended to replace the much weaker and even more dangerous Basic mechanism.* [14]

One of the common attacks is e.g. TCP session hijacking.

3.6.2 Web authentication weaknesses

Many big electronic mail providers now offer the possibility of accessing their mailbox using the web to their users. Unfortunately, the authentication mechanisms suffer from the same problem POP3 does. Username and password are often transmitted in plain text, neither encrypted nor protected by a SSL connection. For example, gmx.net, one of the biggest mail providers in the German-speaking region, transmits the username password combination within a HTTP POST request as following:

```
?user=wallerberger&pass=passme
```

As many users use this method of checking their mailbox, this lack of any security measures makes improving security measures in mail protocols absurd. A possible solution to this problem is discussed in Ch. 5.1.3.

3.6.3 SMTP authentication weaknesses

SMTP simply does not provide any possibility for authentication. It must trust any user that it sending messages. Even mail providers do not have the chance to check via SMTP if the sender really holds the mailbox he pretends to do.

Therefore, e-mail providers developed a SMTP/POP3 authentication hybrid to avoid beeing totally open to these address fakes. Before sending a message via SMTP, a user must firstly authenticate via POP3 within a period of time before. Though this may seem quite secure on the surface, on closer examination, however, this method inherits all problems of POP3 authentication and arises some new problems: any user within the same network or able to fake the IP address is able to send mails within that period of being authorized.

4 SECOL Mail server architecture

4.1 The problem

SECOL stands for Secure Communication under Linux, a complete Mail Server. The following goals were defined at the beginning of the year:

- Implementation of a complete network mail server
- "Pessimistic" filtering of potentially virus-containing contents
- Usage of a "pessimistic" authorization technique in order to deny spam messages completely.
- Inclusion of a configurable mail surface.
- Inclusion of all important mail fetch protocols (POP3, SMTP)
- terminal-based configuration surface.
- complete embedding of data base.

4.2 General program architecture

The software was designed partly as a two tier and partly as a three tier-architecture. This split in principles arose from the different protocols that are provided by this server. In some cases, the server also provides the view tier, and, therefore, fulfills the requirements of a three tier-architecture. In most cases, the server only provides the protocol implementation and data storage and is therefore a two tier-architecture.

The server itself consists of three main modules (Fig. 6):

- *data complex*: This complex provides access to the various data sources. All data except message data is stored within a data base to provide quick and powerful access to the using modules. Messages are stored within the file system to provide flexible access and minimized data base size. These two components are connected via a single data management interface.

- *libraries complex*: This complex provides logical access components for the data base as well as conversion functions for the huge amount of conversions necessary to interpret data correctly. These libraries are critical to the server module.
- *server complex*: This complex provides the server controller, which enables and controls all other complexes and the protocol controllers, which listen for connection and delegate the work to protocol handlers.

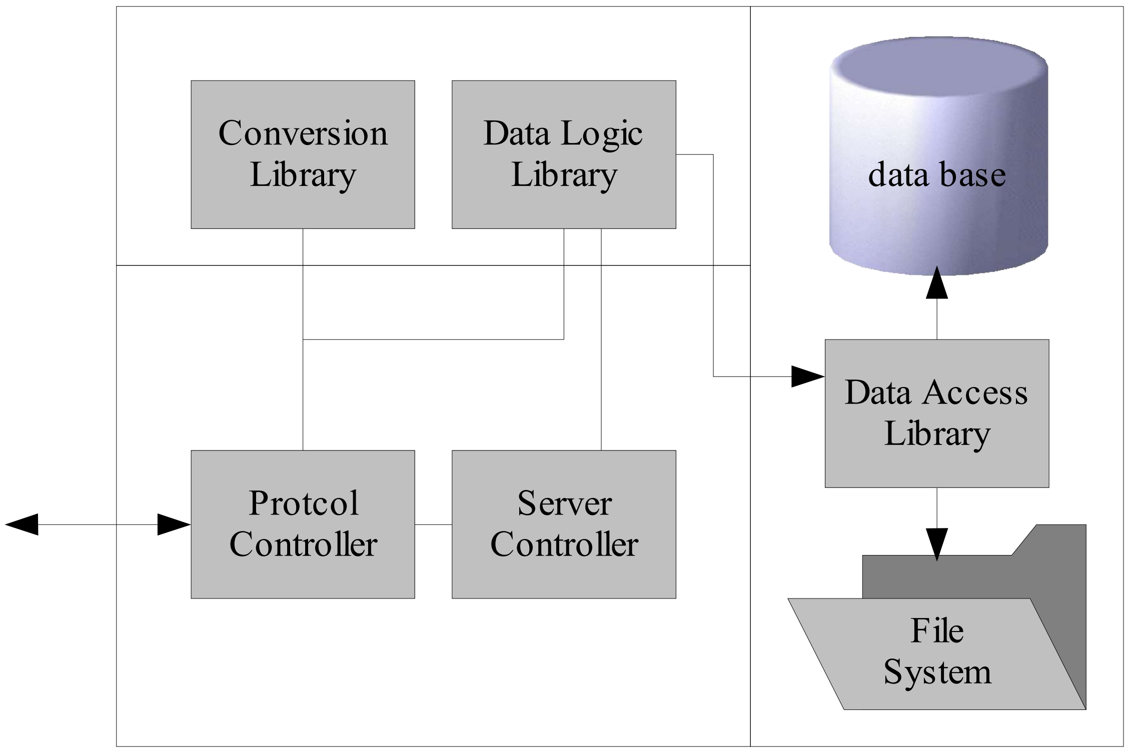


Fig. 6: SECOL Module Architecture

4.3 *Data complex architecture*

4.3.1 General views

In the beginning of the project, the data base was intended to store all data that has to do with the mail server. This idea was dropped, because of seveal reasons:

- *table size limitations*: The data base I intended to use was MySQL. This data base is in it's current version 4.1 not able to store table sizes greater than 4 giga bytes.
- *Not needed*: For computing messages, the data base was not needed. It would have filled the data base unnecessarily.

- *Advantages concerning Linux*: When storing mails in files, there is an advantage when using Linux built-in tools. In fact, Linux daemons work with files that only have to be moved and changed to be appropriate for the mail system.
- *Easier deployment*: Because Linux' mount system allows easy inclusion of remote content, storing messages in files allows easier management of a big amount of mails.

Therefore, another option was preferred – storing messages within files and leaving the operative overhead to the data base. This has also the advantage that the data base remains fast and is limited to really needed data.

Between the two data storage systems, a data access library, written in C++, was placed to secure a well-defined and secure interface to access data is used by the whole server (s. Fig. 6). This interface is mainly used by the data logic library (s. Ch. 4.4).

4.3.2 Views on the data base

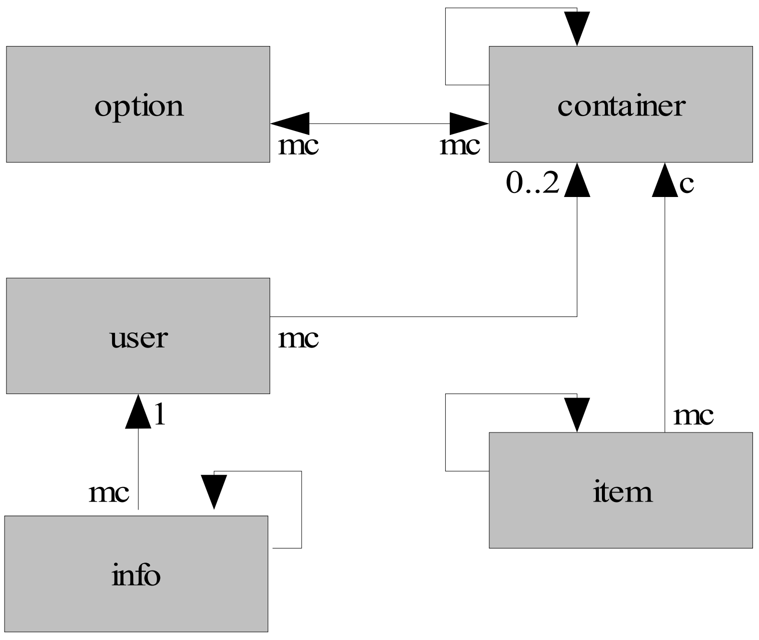


Fig. 7: Entity Relationship Diagram

The entity relationship diagram (s. Fig. 7) only uses 5 main entities, though it does not show all entities that were included into the data base. This model was intended to save not only data for a mailing system but also for a communication platform and similar purposes. An extension should be easy due to this architectural choices.

4.3.3 Users entity

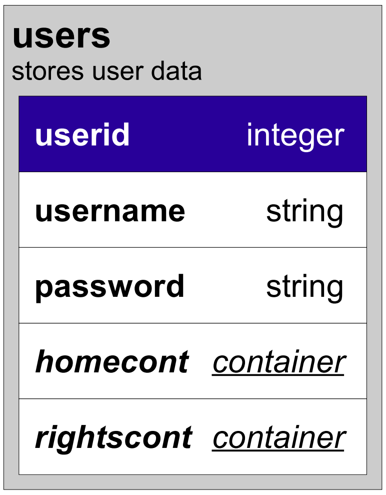


Fig. 8: Users entity

The *users entity* represents the central entity for all user-specific data, but it doesnot store all of this data (Fig. 8). This is done by the info entity, which will be described below.

User names are embedded within a "unique" constraint. This assures unique user names even on data base level. An additional userid was created, because it shrinks the amount of space needed in the info entity. The user entity must store the password in a non-hash value. Later, this may be replaced by an synchronic encryption to avoid that other local programs may read out the passwords.

Moreover, the user entity has a 2 - n relationship to the container entity. As rights logic is done within the container entity, these pointers mark entry points for the user into the container hirarchy. The homecont represents the container the user has it's addresses, the rightscont defines the highest container the user can modify (s. also Ch. 4.3.6).

4.3.4 Info entity

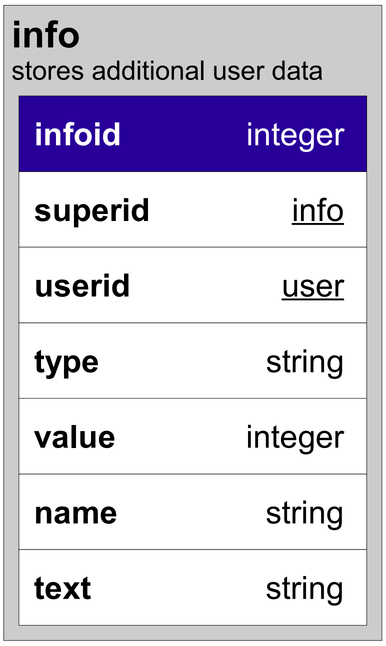


Fig. 9: Info entity

The *info entity* is a sub data base able to store all user-specific data that may not be stored within the user entity. By using this way of representing data, it is possible to extend the data used without changing the data model completely or overloading the user entity (s. Fig. 9).

As a result of the fact that e.g. contact data require a hirarchy, the entity is recursively designed using the superid field. The two-characters' type field then splits the entity virtually in sub-entities, which has nothing to do with each other. Some defined types are `'C'` for Contact data and `'SE'` for session keys.

The value, name and text fields have different meanings for each types. The value field is intended to store numeric values or keys. The name and text fields are also to be a key/value pair. This data structure enables maximum flexibility for storing user data. One may argue that the info entity may be dropped and this data be stored within the preferences entity. This option was not chosen according to the following security consideration: Clients may add entries it the info table, but may only change the more important preference entries.

4.3.5 Container entity

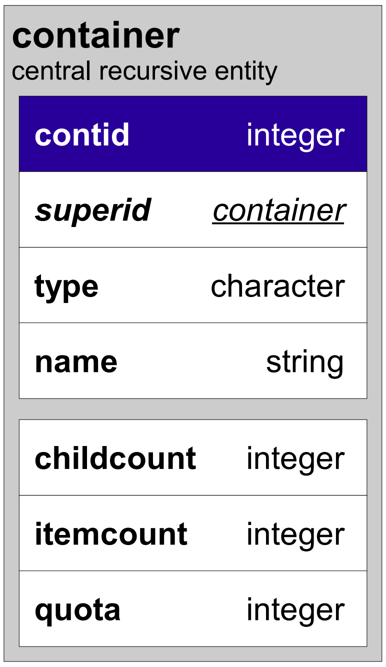


Fig. 10: Container entity

The *container entity* is the central entity of the system (s. Fig. 10). It has a huge amount of functions that are implied by the data model. A container itself is a depot where sub-containers and items (messages) can be placed.

Containers have three cumulative fields that must be updated recursively at every change of the sub-containers or items. Field childcount summarizes the number of all sub-containers, itemcount the number of items in these containers and, finally, quota, which stores the total size of all items and containers that are within the scope of the current container.

Containers are splitted into seven operative groups that are marked by the type flag. Fig. 11 shows the IS-A structure of these types: a container can be either a directory or a folder. To go into furhter detail, the following types are defined within the system:

- *directory*: system-wide containers usually created by administratiors A directory cannot contain items and is created on system level; a user usually cannot change settings on directory level.
  - *module*: The highest abstraction level of containers. Modules are intended to define kind of preference spaces for users. Modules can be recursively connected and may also contain user directories. Type flag `'M'` is assigned to modules

  - *user directory*: A home directory for a user. By using these containers, settings can be done for exactly one user. Normally, a home-dir entry is made for each user directory. A user directory may contain address directories and folders and has been assigned type flag **`'U'`**.
  - *address directory*: For every internal or external e-mail post box, an address directory is created. This directory must at least contain one inbox, one outbox and ont sent folder (s. below). Moreover, address directories enables special settings such as POP3 delivery service a.s.o. and has been assigned type flag **`'A'`**.

- *folder*: Usually, a folder is created by a user. These containers may hold items and are itended as a logical classification for this messages. Folders itself may only contain folders and messages. No directories are allowed within these objects!
  - *default folders*: These folders are only valid within address directories and have defined roles for the worker processes. For this reason, default folders may not be created, altered or deleted by the users. Default folders may only contain custom folders.
    - *inbox folder*: All incoming messages are automatically delivered to this container. Type flag **`'I'`** is assigned to inbox folders.
    - *outbox folder*: Messages placed in this folder are collected by the worker processes and sent. Type flag **`'O'`** is assigned to outbox folders.
    - *sent folder*: Messages that were sent successfully are moved to the specific sent folder for archivation and further processing. Type flag **`'S'`** is assigned to these containers
  - *custom folders*: These folders can be created, altered and removed by the user. They can contain further custom folders if preferred. These containers may occur within folders, addresses or user directories and have been assigned type flag **`'F'`**.

The container objects have to been checked for consistency regularily due to the redundant fields they contain. Moreover, all configuration is assigned to containers. This data also has to be checked.

The “System” - container is obligate, it must have ID 0.

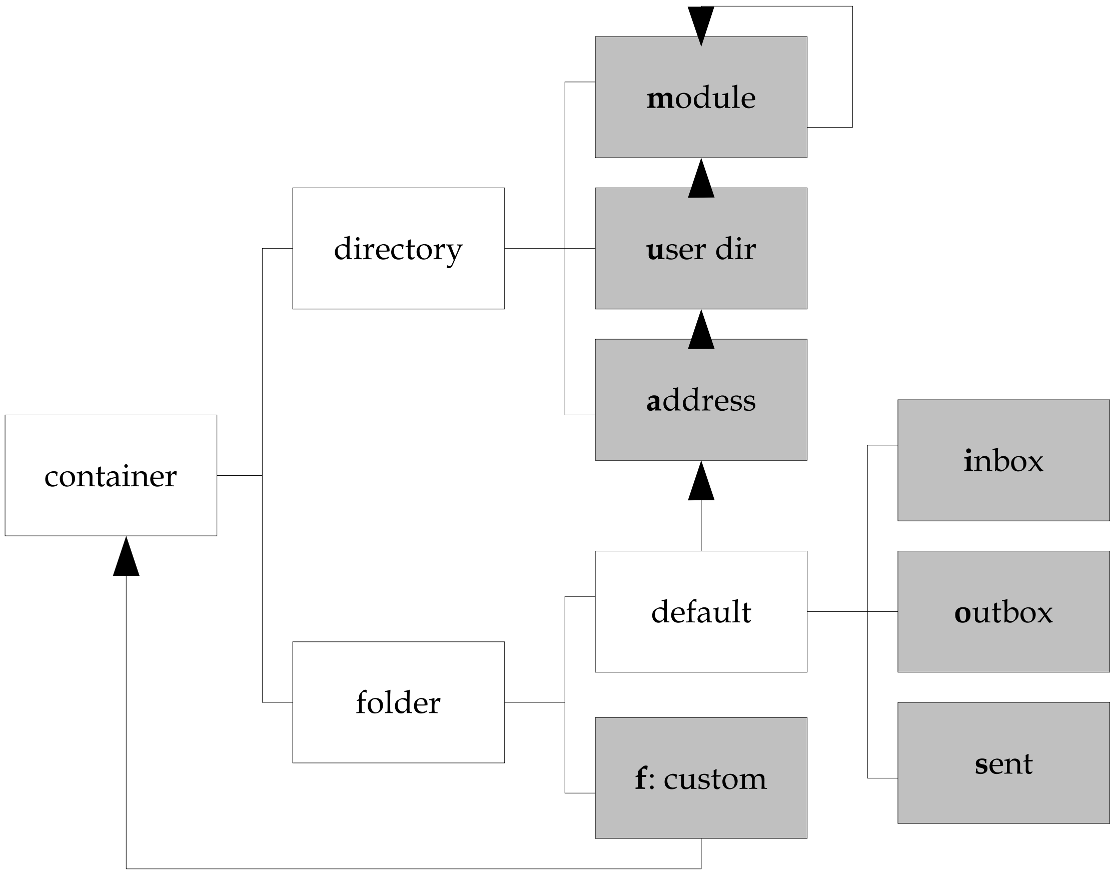
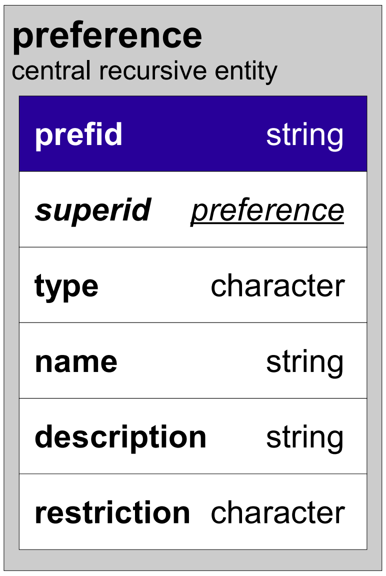


Fig. 11: Full ERD of the container entity (provided by type field)

4.3.6 Preference and value entity

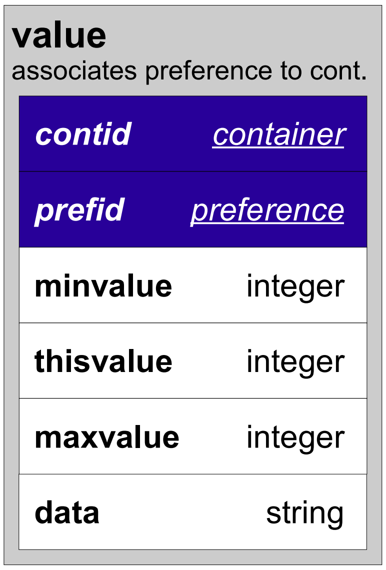
The *preference entity* stores information for the options that can be set for containers. Each preference is identified by a two characters' string, which is used by the value entity. Preferences may be recurisive to support preference groups. By entering the field restriction, a container type restriction may be given. This is rather used to give a better overview than providing a security measure.



preference
central recursive entity

prefid	string
superid	*preference*
type	character
name	string
description	string
restriction	character

Fig. 12: Preference Entity



value
associates preference to cont.

contid	*container*
prefid	*preference*
minvalue	integer
thisvalue	integer
maxvalue	integer
data	string

Fig. 13: Value entity

The type flag may be one of these characters: 'B' for boolean values, 'N' for numeric values, 'S' for strings and 'G' for groups that do not have any functional use.

The value entity is the associative entity that realizes the m to n relationship between the entities preference and container. It, therefore, assigns a preference value for a container. If the value is numeric or boolean, the current value is stored within the field thisvalue, in every other case it is stored within the data field.

By definition, there must be a value for every preference for the System container (ID 0). A subcontainer may change a numeric value between the interval (minvalue, maxvalue) (if given) and is also allowed to set the borders sharper. A string may only be altered by a subcontainer if both minvalue and maxvalue are not given. A new value entry is only generated if something changes compared to the super container. The user also may only alter values for container that are below its rightscont container (important!).

For this reason, one of the most important procedures is the determination of the actual value for a container. If there is no value entry for the desired container, it determines the super-container and checks whether this container provides a value. For this reason, the System container *must* give a value.

4.3.7 Item entity

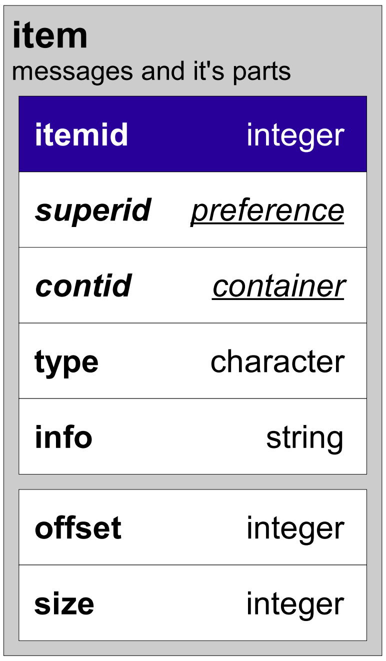


Fig. 15: Item entity

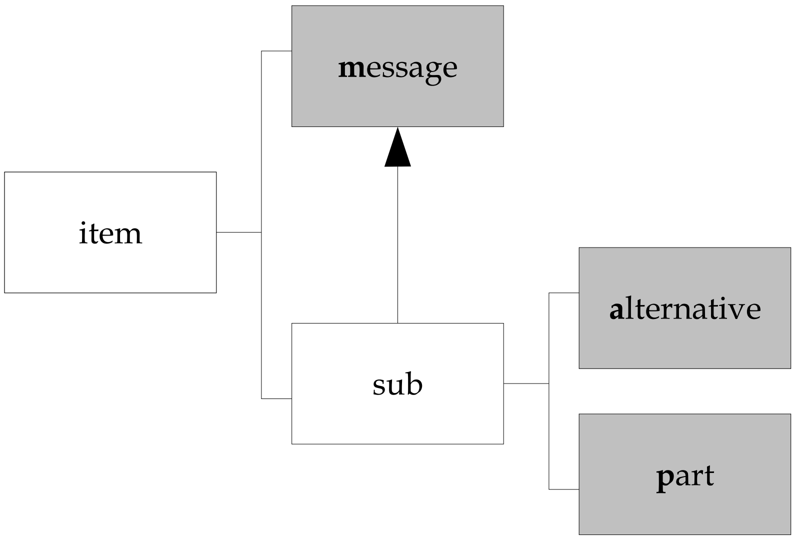


Fig. 14: Item full ERD

The *item entity* stores the messages but does not store the contents of messages, but only provides entries which link to a certain position in the file system (s. Ch. 4.4). An item entry can be either a message or a part of a message. If it is a message, it does not have a parent and is stored within a container. If it is a part of message or an alternative, superid stores the message the part belongs to. Moreover, offset gives the position within the item where the part starts with a length given in the size field.

4.3.8 Views on the file system

In the preference entity, there is also a preference named storage `'SG'`. This preference can be set for each container individually. It stores the absolute path to the mail storage area. For example, when SG is given for a module container with '/mnt/networkstoreA/', all mails of users in this module are stored within this directory. The filenames are standardized: they are always **`SECOL<itemid>.item`**.

4.4 *Library complex architecture*

4.4.1 Data Logic Library

The data logic library defines classes that are wrapped over a record of an entity. For example, a item class was created. The general form of these classes is:

```
class *** : Component
{
public:
   ***(DataBase* conn);

   int fill();
   int query();
   int update();

   /* Fields for this class following */

   /* ... */
}
```

For every entity, there is one class providing members for representing the fields in the data base. The fill function retrieves one record and stores it into the current instance. The query function returns an array of Components with all rows retrieved. Both functions check for fields that were previously entered and uses them in the WHERE – clause of the query. The update function updates the data base if the fields have been changed. Below, there is an code example:

```
Container record(conn);

(record.name = new char[20]) = "System";
if( record.fill() == 1 )
{
    record.name[3] = 't';
    record.update();
}
else
   printf("Error");
```

Moreover, the Data Logic Library provides function for determining things that, usually, would be done using stored procedures. As MySQL does not provide these, this logical functions were packed into a library.

4.4.2 Data Conversion Library

This library provids functions converting data representations into another forms. This is important, because the web works with many representations of data. The following forms of encoding and decoding are supported:

- *HTTP Query Strings*: Encoding and Decoding of these strings is especially important for the CGI server module.
- *Base64 Entites*: As most of the mail attachments use Base64 encoding for transferring binary data, the server must be able to decode that to send it to a browser or encode to send via mail.
- *MySQL strings*: Here, special attention is required. By manipulating these MySQL strings, users may have direct access to the data base. Therefore, encoding of string ending characters is required.
- *Protocol elements*: Encoding and decoding of certain requests and responses of the POP, SMTP and IMF is necessary for talking these protocols correctly.

4.5 Server complex architecture

4.5.1 General views

The architecture of the server complex can be displayed best with a process diagram (s. Fig. 16). As the legend shows, these processes are dividedinto two main groups: the persisent processes ("controllers") that are present all the time the server is running and the generic/temporary processes ("handlers") that are generated for doing special tasks and aborted then.

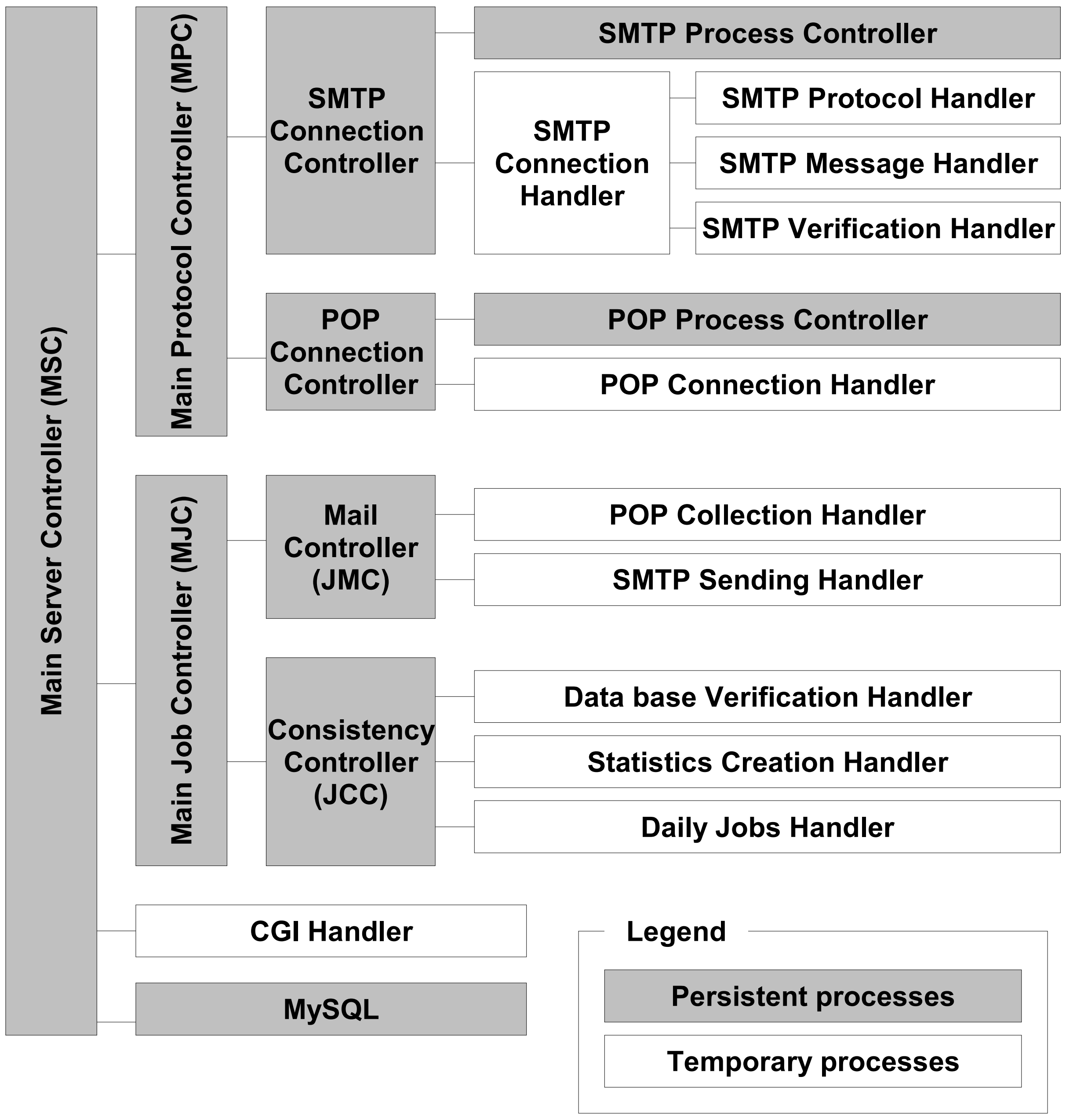


Fig. 16: Process Diagram

4.5.2 Main Controller Processes

The Main Server Controller is the process started by the user. The MSC does the following on start-up:

- Initializion of the data base, data base parity checks
- Testing of the CGI scripting
- Setup of the Shared Memory segment
- Startup of the Main Job Controller MJC and the Main Protocol Controller MPC

The main protcol controller queries for the implemented protocols and starts the according protocol controllers. These controllers then listen for connections at the protocols' port and call protocol handlers when they got a connection.

The main job controller starts the mail-controller, which is responsible for sending mails via the SMTP protocol by using the sendmail command and for the gathering service via POP3. Secondly, the consistency controller is started. This process starts processes, which check the consistency and integrity of the data base and the file system. Moreover, these processes also generate activity statistics and perform daily jobs (deletion of old mails etc.).

5 New Solutions

5.1 *Extended Mail Transfer Protocol XMTP*

5.1.1 Overview

This protocol is intended as a follow-up protocol to obsolete SMTP. It provides security measures and measures for sender IP detection and structured as simply as SMTP. Moreover, it leaves out some of SMTP's deprecated commands. This protocol would be a good solution for most of the SMTP's urgent problems.

A XMTP connection works as following: The client connects to the server at port 25025 and sends the **`HELLO`** command. If wanting to send an e-mail, the client must first use the **`AUTHENTICATE`** command to verify all sender addresses he wants to use. The user may accept the address instantly by sending **`SUCCESS`** or may challenge the client by using the **`VERIFY`** request. When all users are verified, the client may send multiple mails using the **`MAIL`** request or close the connection via the **`QUIT`** command.

5.1.2 XMTP Verification

The **`VERIFY`** command is the central element of the protocol. Both the client and the server may send this request. Usually, the server will send this command to assure the identity of the client. The client can use this request for the back-tracking mechanism described below. Possible answers to this request are:

- **`SUCCESS`**: This command totally confirmes the identity of the requested address. This answer may only be given by a server that is sure that this address is valid and hold by this server. A client *must not* give this answer.
- **`FAILURE`**: If the address cannot be verified or is definitely not valid, this return must be sent. This may happen e.g. when a server knows that a message from this address never passed it's XMTP/SMTP.
- **`ASK`**: A server/client may give this answer, followed by the domain name/IP address of a XMTP server, when it only redirects a message and does not know if the address is valid. It then must give the last server that used this address. The other communication partner may then ask this server for verification (= back-tracking).

- **AUTHENTICATE**: This response may only used by a client to verify addresses. It is usually followed by authentication information according to the verification method given by the request partner.

The sender of the request may then accept the verification (**SUCCESS** command), deny it (**FAILURE** command) or may challenge the other communication partner again by using another **VERIFY** command. By using this method, multi-challenge authentications (Kerberos etc.) are possible. A server *must* store all users that are verified within one session.

The **AUTHENTICATE** command may also be used to force the server to send a verify command.

5.1.3 Sample communication

Before going into further detail, a sample communication should illustrate the principle of the protocol. Note that a double slash "//" starts a single-line C++-style comment. We assume the following situation: wallerberger@badguy.ru connected to **xmtp.badguy.ru** to send a mail to **nice@goodguy.us**. The XMTP did not send the message directly, but chose to use **mail.publicrelay.de** as a mail relay.

Step 1: Client wallerberger@badguy.ru (C) tries to send it's mail via server xmtp.badguy.ru (S):

```
C: HELLO
S: SUCCESS
C: AUTHENTICATE wallerberger@badguy.ru
S: VERIFY wallerberger@badguy.ru method="default"
C: AUTHENTICATE wallerberger@badguy.ru pass="nicetoall"
S: SUCCESS
C: MAIL
S: REQUEST Size UID
C: MAIL Size=39388 UID=<stephen.franks@gmx.at>
S: REQUEST Mail
C: MAIL Text="blah blah blah
blah blah blah
blah \" blah blah
\\ blah"
S: SUCCESS
C: QUIT
S: // 1 message/39388 bytes sent
S: // Terminating connection
S: SUCCESS
```

Step 2: Client xmtp.badguy.ru (C) tries to forward mail to server mail.publicrelay.de (S). This step was left out.

Step 3: Client mail.publicrelay.de (C) tries to deliver the mail to goodguy.us.

```
C: HELLO
S: SUCCESS
C: AUTHENTICATE wallerberger@badguy.ru
S: VERIFY wallerberger@badguy.ru method="default"
C: ASK mail.publicrelay.de
S: // please wait while performing back-tracking

Server (C) now opens connection to mail.publicrelay.de (S)

C: HELLO
S: SUCCESS
C: VERIFY wallerberger@badguy.ru
S: ASK xmtp.badguy.ru
C: QUIT
S: SUCCESS

Server (C) now opens connection to xmtp.badguy.ru (S)

C: HELLO
S: SUCCESS
C: VERIFY wallerberger@badguy.ru
S: SUCCESS
C: QUIT
S: SUCCESS

Server now continues first connection...

S: // back-tracking complete (2 servers/29 ms)
S: SUCCESS
C: MAIL
S: REQUEST Size UID
C: MAIL Size=39388 UID=<stephen.franks@gmx.at>
S: REQUEST Mail
C: MAIL Mail="blah blah blah
blah blah blah
blah \" blah blah
\\ blah"
S: SUCCESS
C: QUIT
S: // 1 message/39388 bytes sent
S: // Terminating connection
S: SUCCESS

Message is now delivered.
```

5.1.4 XMTP Commands

In general, XMTP commands are identified by the first word. Optionally, a object may follow directly. This is e.g. an address in the authenticate command. Additional parameters may be given as a key/value pair seperated by an equal sign. If the parameter value contains whitespaces, line breaks, qoutation marks, backslahes or equal signs it must be masked and enclosed within quotation marks. Parameters may be seperated by any linear whitespace (for definition see Ch. 5.1.5).

HELLO command

• Syntax:	**`HELLO`**	
• Parameters:	none	
• Execution:	Client, once at startup	
• Responses:	SUCCESS	Service waiting for further commands
	FAILURE	Service is not ready or the client is not accepted

The hello command is the first command a client *must* send to a server. It greets the server and is requesting execution of further commands. In the case of a negative response, the connection should be closed immediately

VERIFY command

• Syntax:	**`VERIFY`** ***`email-address`***	
• Parameters:	[method]	stores the way of authenticating
	[step]	stores the current step number in the auth process
	uid	stores UID of message to authenticate
• Execution:	Client and server	
• Responses:	SUCCESS	Server verifies address
	FAILURE	Server/client is not able to verify address
	AUTHENTICATE	Client tries to authenticate address
	ASK	Server/client refers to previous sender of the mail

The verify command is described in Ch. 5.1.2 in further detail.

AUTHENTICATE command

- Syntax: **`AUTHENTICATE`** ***`email-address`***
- Parameters:

uid	stores UID of message to authenticate
else method-dependent	

- Execution: Client, in response to a VERIFY or to force server to send VERIFY
- Responses:

SUCCESS	Authentication successful
FAILURE	Authentication failed
VERIFY	Another challenge for authenticating client

The authenticate command is described in Ch. 5.1.2 in further detail.

MAIL command

- Syntax: **`MAIL`** ***`email-address`***
- Parameters:

[size]	Stores total mail size
[uid]	Stores unique mail ID
[header-**]	Stores value of IMF header **
[mail]	Contents of the mail, may only be sent when requested

- Execution: Client
- Responses:

SUCCESS	Mail sent successfully
FAILURE	Mail could not be sent
REQUEST	Request for another parameter(s)

With the mail command, the client may drop a mail at the server. The mail itself is stored within the mail parameter which may not be sent until the server requests for it using the request command. By using this request technique, the server is able to check if the size is acceptable and do other verifications before the client is allowed to send.

REQUEST command

- Syntax: **`REQUEST`** ***`param1 [param2 ...]`***
- Parameters: none
- Execution: Server, in response to MAIL command

- Responses:

MAIL	Mail command providing these parameters
FAILURE	Parameters cannot be provided

The request command is used for retrieving additional parameters for a mail from the client. It is the only way to get the mail-parameter, which contains the mail itself.

QUIT command

- Syntax: `QUIT`
- Parameters: none
- Execution: Client
- Responses: SUCCESS Server closes connection.

The success and the failure command were not issued.

5.2 *Reverse Lookup Authentication*

5.2.1 Overview

The reverse lookup provides a quite secure mechanism to determine the origin of an e-mail also via SMTP. This can be used for authenticating the sender of a mail by returning a mail which contains a not machine-readable verfication code the client then has to enter.

5.2.2 Authentication Procedure

Fig. 17 visualizes this process of verification: firstly, the original message sent by the source is stored within a temporary folder (1). It is important that the mail comes directly from an address within the source domain and was not resent by any mail relay, because the mail server then performs a DNS reverse lookup procedure to verify that the server is authorized to use the given mail domain (2). If the reverse lookup successes, the server returns a verification request (3). This request contains a link, which leads the sender to a web page. He then has to copy a non-machine readable alphanumeric combination into a textfield to secure that the e-mail address is not generic (4). If this procedure successes, the mail is delivered (5).

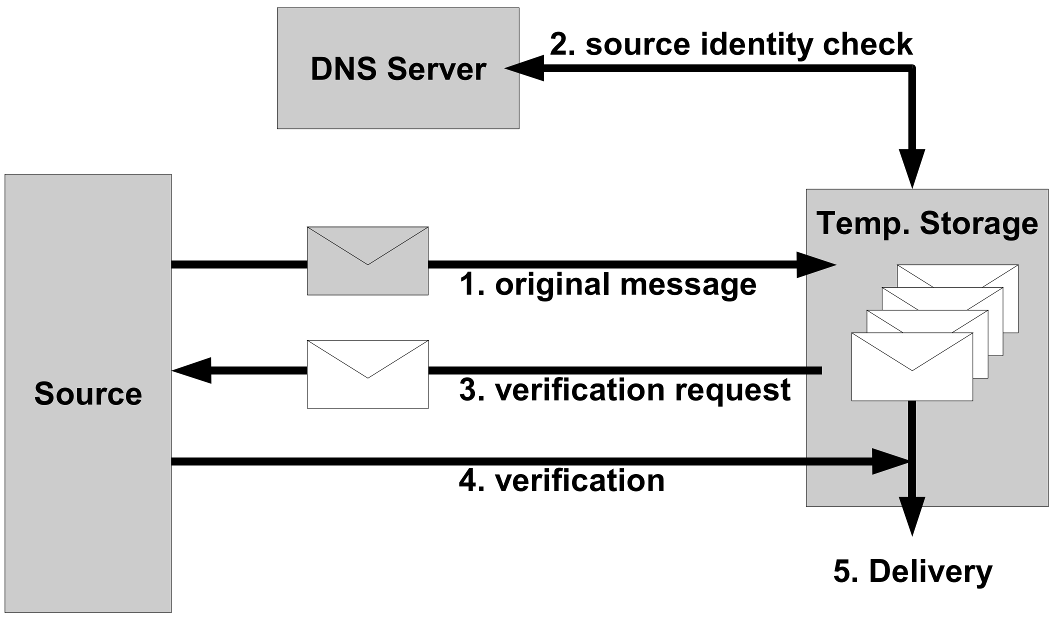


Fig. 17: DNS Reverse Lookup authentication

5.2.3 Weaknesses of the DNS Reverse Lookup authentication

Due to its strict character, the DNS Reverse Lookup authentication has an array of disadvantages and is shrinking the freedom of mailing drastically. The most important weaknesses are:

- *Insufficient implementation*: Only the bigger server already implemented the Reverse Lookup procedure. Moreover, a server can only implement Reverse Lookup when it's network provider also implemented an entry and linked to this server.
- *No mail relays*: As the server uses the SMTP client's IP for reverse lookup, the use of mail relays is not possible, because the mail relay usually doesnot hold the domain of the source address it forwards.
- *No alias domains*: For the same reason, sending mail on behalf of another address not on the same server. Some automatically generated mails, therefore, are blocked due to the reverse lookup failure.
- *Huge temporary storage*: Because the sender must confirm it's identity, the server must hold mails in it's temporary storage for quite a long time. This requires an enourmous amount of disk space. Moreover, a DoS attack[6] by flooding the server with unauthorized messages is also possible.

6 Denial of Service attacks try to crash a service by exhausting the servers capacities (processor, bandwidth, TCP/IP buffers, RAM and/or disk space)

- *Manual authentication required*: The sender must confirm at least one mail per address he wants to send to. This results in a additional overhead. Moreover, authomatic mails are rejected in any case.

5.2.4 Implementation in the SECOL server

The SECOL server made the following additions to the reverse lookup authentication:

- A e-mail address must only be authenticated when it is sending it's first message.
- After that, it may send 20 messages a day without authenticating again. If it exceeds this quota, it must authenticate again to get additional **`lastqouta-4`** (16, 12, 8, 4) messages. For this reason, it does not pay for spammers to authenticate, because they can only send a maximum of 20 messages a day before they must authenticate again.
- To avoid DoS attacks, the buffer for unauthorized messages may only hold one message per address. Moreover, the buffer will only hold as many messages from one IP address as already registered users for this IP address, increased by 1.
  For example, if mail.gmx.at holds the IP address 19.39.29.1 and may place 3 messages within the temporary message buffer, if two e-mail addresses for 19.39.29.1 are already registered.
- The module administrator may configure the number of messages going to this module from one IP class C net address a day, before it requires addresses to sign up. This may be varying depending on the number of clients. By default, the SECOL mailing uses the function given in Eq. 1 for calculating the number of non-authenticised mails N (s. also Appendix).

$$N(x)=\frac{\ln(x)-\ln(16)}{\ln(2)} \qquad \text{(Eq. 1)}$$

- Optionally, the DNS reverse lookup is replaced by a similar technique that uses more disk space: For each mail address, the IP address, which had the latest connection, is stored. If this IP address is changed for an address, mails are rejected until they are authorized. Only the first 24 bits are relevant for this comparison.
  For example, if wallerberger@gmx.at connected via 11.302.13.11, this is stored within the system as **`wallerberger@gmx.at@11.302.13`**. If a spammer wants to send mails via wallerberger@gmx.at using IP 17.3.1.9, he first must authenticate.

5.3 *JavaScript Client*

5.3.1 Overview

For the management of one's personal mail accounts as well as the administration of the complete mail server, a JavaScript client was developed. The special thing about this JavaScript client is the fact that it communicates via an own HTTP-based protocol talking with a CGI script by default or any other server implementing this protocol (PHP etc.). Moreover, as the surface is generated dynamically via JavaScript, the interface design is configurable completely freely and no structural element have to be down-loaded from the server.

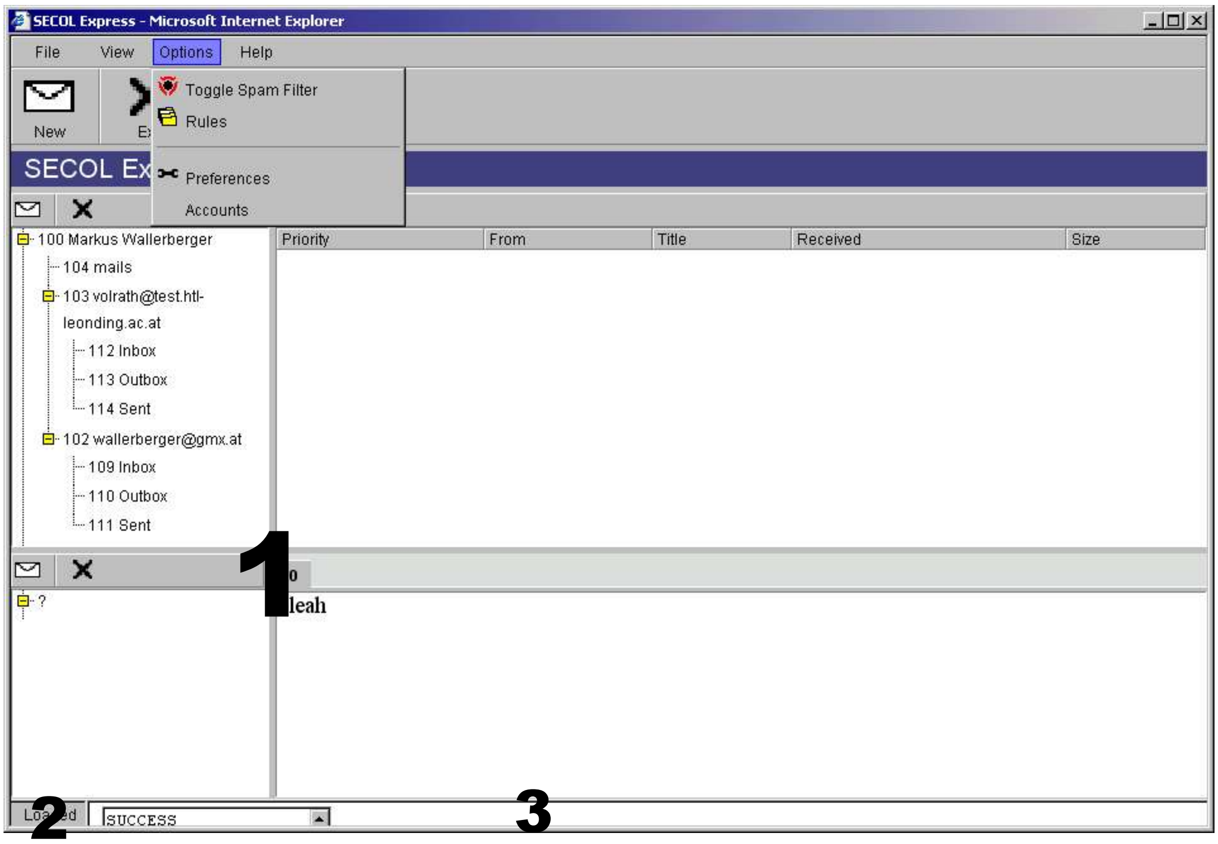


Fig. 18: JavaClient Surface

5.3.2 Client Design

The client is splitted up into three frames (s. Fig. 18). The biggest of these frames is the *interface frame* (1), which provides the graphical interface and the controls for user interaction. The other two frames form the status bar at the bottom of the window, but provide all the functionality of the client. The *controller frame* (2) updates the interface on changes and processes inputs from the user. In addition, it interacts with the *protocol* *frame* (3). This frame contains a HTML form. This HTML form is filled with commands and posted to the server. The server then returns the commands and forces the controller to interpret and update the interface.

The client contains the following libraries:

- *mask.js*: Provides a complete library for generating Windows-like controls completely JavaScript based. Currently, it is able to generate expandable trees, menus with submenus, toolbars with roll-over effects, lists and tab pages. It was hard work to get this library running.
- *func.js*: Provides general functions for input and output. Moreover, it provides a built-in error message library which is very easy to use. Finally, it provides special string processing methods for command interpretation.
- *main.js*: Provides the functionality of the client. Responds to all events a user takes, form protocol requests and analyses protocol responses. Furthermore, it iniatializes the client on it's start-up.
- *md5.js*: Provides an open-source implementation of the MD5 message digest algorithm. This is needed for the authentication mechanism (s. Ch. 5.3.4)

5.3.3 Client Protocol

This chapter will give a brief overview over the protocol elements used within the JavaScript client (s. Tab. 3). It is not intended to be a complete reference to this commands.

Command	***Description***
VERIFY	Used for identifying the user (s. Ch. 5.3.4)
FOLDER	Used for requesting/updating list of folders under certain folder
ITEM	Used for requesting/updating items of a folder
OPTION	Used for requesting/updating values for folder preferences
INFO	Used for retrieving user/session information

Tab. 3: JavaScript Command list

5.3.4 Client Authentication

Fig. 19 displays the authentication algorithm used by the JavaScript client. Unlike most of the web surfaces (s. Ch. 3.6.2), the JavaScript client provides a quite secure authenticate technique which is much like the CRAM-MD5 (s. Ch. 3.6.1) algorithm, but more secure.

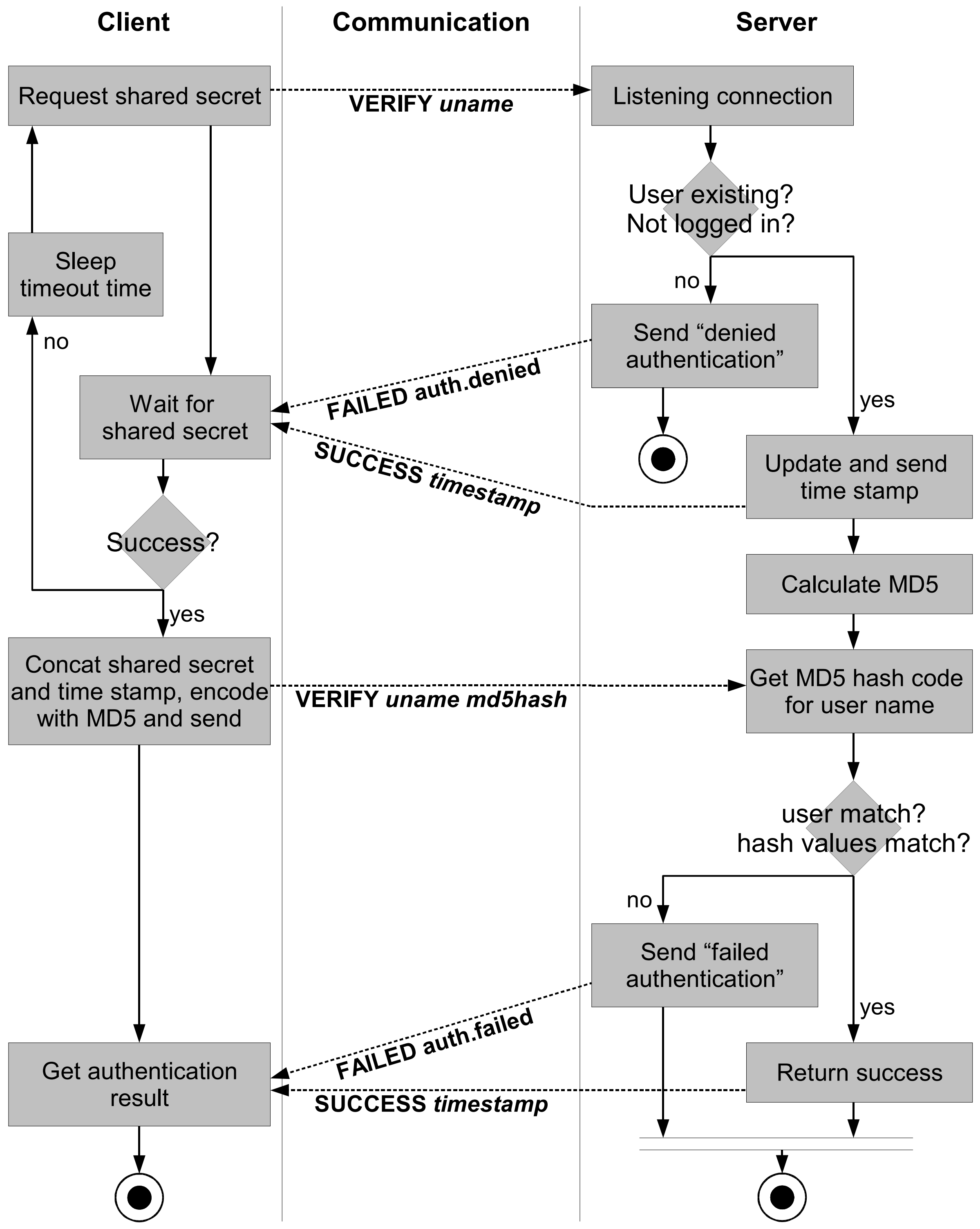


Fig. 19: JavaScript Authentication Process

6 Conclusion

As these lines are written, Microsoft founder Bill Gates announced some weeks ago, "specialists are working on a method to eradicate the spam problem within 18 month". This paper showed that this approach is promising for the future. Though the Internet does not provide possibilities for complete eradication yet, it showed that the most serious flaws, anonymity and cheap deployment, can also be faced by using present methods.

Fighting the symptoms will not pay in the long run. Although the number of filtering techniques and tools are growing exponentially, spammers always find ways to escape these mostly statistical filters. Seemingly, it is an endless head-to-head competition. As mentioned, it could be ended by deprieving spamming it's right to exist.

Maybe this should also be done at the virus frontier, where this game is played for several decades now. It may be worth thinking about reducing mail to it's original and primary use: a fast and powerful way of asynchronous textual communication. This would make it impossible to deploy viruses via electronic messaging.

The weaknesses of the protocols, rising major security and privacy issues described in this paper, must be fought in any case. Changing the Internet protocol structure by adding some new feature to certain protocols would make anti-spam tricks needless. Unfortnately, the slow implementation of IPv6 showed that changing structure on the Internet isn't as easy as one would imagine. The growing significance of this problem and a quick implementation of the global players, however, could make this change possible. It is a fact that we cannot work with these "protocol artefacts" any longer.

It is needless to underline the importance of electronic mailing and therefore the significance of fighting threats to e-mail: In many companies, it is the #1 medium for sharing ideas, data and work in general. Mailing server, therefore, are providing the clear majority of shared workspace platforms – not the special shared workspace tools! For this reason, extended mail servers, providing additional functionality for sharing data will probably be the working platforms of the future.

7 Bibliography

[1] Hardy, Ian R.: "History of ARPANET E-Mail: A History Thesis Paper". University of California at Berkeley, 1996. Internet:http://server.berkeley.edu/virtual-berkeley/email_history, Accessed .

[2] Myers, J., Rose, M.: "Post Office Protocol - Version 3 (RFC 1725)". Dover Beach Consulting Inc., 1994. Internet:http://www.ietf.org/rfc/1725.txt, Accessed 2004-03-01.

[3] Crispin, M.: "Internet Message Access Protocol (RFC 1730)". University of Washington, 1994. Internet:http://www.ietf.org/rfc/1730.txt, Accessed 2004-03-01.

[4] Freed, N., Borenstein, N.: "Multipurpose Internet Mail Extensions(MIME) Part One:Format of Internet Message Bodies". , 1996. Internet:http://www.ietf.org/rfc/2045.txt, Accessed 2004-03-01.

[5] Wood, Paul: "A spammer in the works: A MessageLabs white paper". MessageLabs, 2003. Internet:http://www.messagelabs.com, Accessed 2003-11-10.

[6] Ungerer, B.: "Eingeengt - E-mail zwischen unerwünscht und unverzichtbar". In: iX Magazine, April 2004

[7] Sophos Inc.: "Spam: A many rendered thing". USA, 2003. Internet:http://www.sophos.com, Accessed .

[8] Klambauer, H.: "Betriebliche Organisation 4. Jahrgang". , 2002. Internet:, Accessed .

[9] Krückl, Karl, Interview by Markus Wallerberger, 2003

[10] Heidrich, J., Tschoepe, S.: "Aussortiert - Rechtliche Tücken beim Einsatz von Viren- und Spam-Filtern". In: , March 2004

[11] Bundeskanzleramt: "Rechtsinformationssystem". , Updated frequently. Internet:http://ris.bka.gv.at, Accessed 2003-10-03.

[12] Tomlinson, Ray: "Interview by Ian Hardy". , 1996. Internet:, Accessed .

[13] Rivest, R.: "MD5 Message Digest Alogrithm (RFC 1321)". MIT Laboratory for Computer Science, 1992. Internet:http://www.ietf.org/rfc/1321.txt, Accessed 2004-03-01.

[14] Franks, J. et al.: "HTTP Authentication: Basic and Digest Access Authentication (RFC 2617)". Northwestern University, 1999. Internet:, Accessed .

[15] Mitnick, K.: "the Art of Deception (die Kunst der Täuschung): Controlling the Human Element of Security". 2002

[16] Stewart, William: "Electronic Mail at LivingInternet.com". Canada, 1999. Internet:http://livinginternet.com/e/e.htm, Accessed 2004-01-03.

[17] Postel, Jonathan: "Simple Mail Transfer Protocol (RFC 821)". Information Science Institute, Southern Carolina, 1982. Internet:http://www.ietf.org/rfc/0821.txt, Accessed 2004-02-01.

[18] Reynolds, J. K.: "Post Office Protocol (RFC 918)". Information Science Institute, Southern California, 1984. Internet:http://www.ietf.org/rfc/0918.txt, Accessed 2004-02-01.

[19] Anonymous: "Kommunikationskanäle". , 2004. Internet:http://de.wikipedia.org/wiki/Kommunikationskan%E4le, Accessed 2004-03-01.

[20] Köck, Gerald: "Betriebs- und Führungspraxis. Präsentation 5. Jahrgang". HTBLA for computer science and business administration, 2003. Internet:www.htl-leonding.ac.at, Accessed 2004-05-01.

[21] Klensin, J.: "Simple Mail Transfer Protocol (RFC 2821)". AT&T Laboratories, 2001. Internet:http://www.ietf/rfc/2821.txt, Accessed 2004-03-01.

[22] Resnick, P.: "Internet Message Format (RFC 2822)". QUALCOMM Inc., 2001. Internet:http://www.ietf.org/rfc/2822.txt, Accessed 2004-03-01.

8 Appendix

8.1 Way of calculating transmission times

Let us take a simple e-mailing example: Let us say, a recipient gets approximately 5 messages a day, the probability of getting between a and b messages a day is given by:

$$\mathrm{Ps}(a \leq X \leq b) = \sum_{x=a}^{b} \frac{e^{-5} \cdot 5^{x}}{x!}$$

8.2 Number of unauthenticized mails

Using the function in Eq. 1 (s. there), the following user tresholds are given for N:

Unauth. mails	User treshold
1	32
2	64
3	128
4	256
5	512
6	1.024
7	2.048
8	4.096
9	8.192
10	16.384
11	32.768
12	65.536
13	131.072
14	262.144
15	524.288
16	1.048.576