An Email Spam Filtering Proxy Using Secure Authentication and Micro-bonds

by

Ariel Lauren Rideout

Bachelor of Science in Computer Science and Engineering,
Massachusetts Institute of Technology 2004

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

June 2005

Author .....
Department of Electrical Engineering and Computer Science
May 19, 2005

Certified by .....
t Miller
ɔervisor

Accepted by ...
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

An Email Spam Filtering Proxy Using Secure Authentication and Micro-bonds

by

Ariel Lauren Rideout

Submitted to the Department of Electrical Engineering and Computer Science
on May 19, 2005, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

The Apuma system described in this thesis was designed and implemented as a novel combination of existing technologies in order to give an email user full control over their incoming email. The innate uncertainty of automatic spam detection creates a tension between the desire to filter 100% of spam, and the need to avoid the loss of legitimate mail. Apuma attempts to solve this problem by combining accept-lists with payment systems and content evaluation. Messages from known senders can be exempted from filtering; combined with intelligent automated management of the accept-list this can eliminate the vast majority of false-positives. Remaining mail can thus be subjected to much more rigorous screening. Finally, first time contact and other special cases can be handled with micro-payments or micro-bonds. Apuma includes a plugin interface that allows any financial, proof-of work, or other desired protocol to be integrated into the Apuma filtering framework.

Thesis Supervisor: Robert Miller

Acknowledgments

I would like very much to thank Professor Rob Miller, my advisor and supervisor during the course of this research. With his constant stream of well-considered questions and ideas, he pushed me to keep finding new ways of improving my work. His experienced advice shaped every aspect of this project.

I owe the entire Chisec group my thanks for always being eager to discuss everything that was going on in any one of our projects. That kind of collaborative spirit reminded me every week why I love engineering.

Simson Garfinkel was especially wonderful. Even when under immense workload, he always found time to discuss discoveries, debate issues, and to share his invaluable knowledge on a wide variety of topics. Many of the ideas in this thesis came out of those conversations. I am especially indebted to Simson for prodding me to learn Python, and for helping me to do so.

I am infinitely grateful to my loving, supportive boyfriend Colin Cross, whose companionship and encouragement kept me going on those days when I most wanted to collapse.

I also wish to thank my roommate Matt Seegmiller for being perpetually and infectiously enthusiastic about software engineering, and always having a good debugging story to tell.

Of course I owe many thanks to every professor and teacher who taught me all the things I needed to know before I could embark on a journey like this project. The bumper stickers says if you can read this, thank a teacher. Well, after finding that I am capable of writing a thesis, all I can say is Thank you, Thank you, Thank you.

Last, and most of all, I wish to thank my family: Ellen Meadows, Steve and Charly Rideout, and Alexis Stember, as well as my grandparents, aunts and uncles, and cousins. My family has made me who I am, they have always been there for me, to celebrate the good times and to pull through the hard times; I would not be where I am today without them.

Contents

List of Figures

Chapter 1

Introduction

Spam is sometimes defined as unsolicited commercial email or unsolicited bulk email. To the user it is simply unwanted email. Because this is the definition of spam that is most meaningful to the user, this is the definition that will be used for this thesis. This is an subjective measurement and thus represents a departure from the more objective measures that impersonal filtering systems use.

The burden of spam has been rising steadily with the success of email, and is now well over 50% of total email traffic according to companies such as MessageLabs[28] who work to filter spam. This spam volume imposes a costly load on network infrastructure, and wastes billions of dollars of workers' time sifting through their inboxes[18]. However, email has become so important that we cannot abandon it, and any solution to the spam problem must balance the need for control with the need to protect the qualities of email that make it unique and wonderful: low cost, easy access, potential anonymity, nearly instant communication, and unregulated free speech.

The arms race between spammers and spam fighters has been in constant escalation, with the spammers evolving and reacting to every new barrier that is thrown up against them. The spammers have an important advantage in the huge size and distributed nature of the internet; without central control, or a unified identification

system, spammers are free to generate an endless supply of fake identities with which to send spam. A known spammer can be ignored, but an unknown sender could be either friend or foe. The only way to be sure is to read the message, and then the spammer has succeeded in getting their message to the user's eyeballs. Spammers also exploit relationships by sending spams with the "From:" address of someone the recipient knows. These social networks are harvested by computer viruses that steal the address books off users' computers. Or more simply, they can generate random addresses in the domain of the recipient and exploit the community relationship.

Automated systems that weed out spam create a tension between the desire to filter 100% of spam, and the need to avoid the loss of legitimate mail. If messages are scored on a continuum, then a threshold line must be drawn to decide what will be treated as spam. The innate uncertainty of spam detection algorithms means that there will be a grey area. Adjusting the threshold is a tradeoff between false-positives and false-negatives. Avoiding false-positives means allowing in more spam.

The Apuma (Anti-spam Proxy Using Micro-bonds and Authentication) system attempts to solve this problem by combining accept-lists[1] with payment systems and content evaluation. A pre-existing relationship between the sender and recipient makes the message not spam, unless the user specifically rejects that sender. Messages from an accept-list of known senders can be exempted from filtering; combined with intelligent automated management of the accept-list this can eliminate the vast majority of false-positives. Remaining mail can thus be subjected to much more rigorous screening. Finally, first time contact and other special cases can be handled with micro-payments or micro-bonds.

Apuma's incoming message proxy is a filtering framework that uses plugins to make decisions about each message. Apuma includes a plugin API that allows any financial, proof-of-work, or other desired protocol to be integrated into the Apuma filtering framework.

[1]Accept-lists and reject- or block-lists are also commonly called white-lists and black-lists. I have avoided that terminology because it's implications are out-dated and offensive.

Outgoing mail passes through a second proxy that adds a digital signature and updates the accept-list to include recipients of the message. Digital signatures allow the recipients to be sure of the sender's identity and to verify that the message content was not altered.

Apuma also includes a graphical interface, allowing the user to view and edit the accept-list, as well as control parameters.

1.1 Philosophy of Apuma

Apuma is an email handling system designed to fit a certain set of ideas about how email should be controlled. This section explains those ideas in detail.

1.1.1 Who should filter email?

The core principle of Apuma's approach to email is that the recipient should be in complete control over what email they receive.

Nobody should be entitled to discard an email except the person it is addressed to. Any email filtering set up by ISPs (Internet Service Providers) that blocks emails, and does not allow the customer to control the settings, violates this principle.

Ideally, senders should be allowed to send any amount and type of email to anyone at any time. ISPs should not block any of their customers' outgoing email. A sudden mass mailing could be a valid newsletter. A message about the price of a stock could be a valid accountant-client communication. Even an email carrying a known virus could be a communication between security researchers. Only the intended recipient should have the option to discard the message; if the sender's ISP discards the message, the recipient never gets the chance to make that decision.

At this point in time it makes sense for ISPs to block floods of outgoing email, since they are at risk for retribution if they shelter a spammer. However, these fear-prodded tactics should be phased out because they stifle email and create a

class system between those who can send mass emails and those who can't. Many home users cannot send "From" their work address or vanity domain while they are connected through DSL or cable modem. While the procedures of the major ISPs appear fairly reasonable, the pressure to have squeaky clean outgoing mail is beginning to have bad effects. Smaller ISPs that cannot afford to take chances are already blocking outgoing messages that appear even the least bit spammy[6].

Blocking personal communications because they matched a few keywords should be considered outrageous, but as long as people hate spam, it will be considered by many to be a necessary casualty. It is imperative that effective, recipient-side filtering be made available before the free-as-in-speech nature of email is degraded completely.

1.1.2 Should sending email cost money?

Spammers spam because the cost of sending a spam is less than the expected return. It is profitable[25]. As long as that is the case, they will keep spamming. There are two clear ways to tip the balance of that equation: increase the cost or decrease the expected return.

By some estimates of the costs and response rates of spam[25], the cost of making a sale by spam is less than $10. The profit on a sale such as a home mortgage, or counterfeit drugs, could easily be five times that much. We can decrease the response rate further by blocking more spam, so that the intended customers never see it. We can educate people about not buying from spam. But ultimately, even if the return rate were divided in half, spamming is still profitable.

So yes, there should be circumstances under which sending an email costs money. Most estimates agree that the monetary return *per spam sent* is some fraction of a penny. The cost of a single penny per email would change the profit balance, and spamming would no longer be worthwhile. The cost of sending an **unwanted email** to a **total stranger** should be at least a penny.

But many legitimate advertisers spend more than a penny to reach a customer.

What if the spammers just get into higher profit markets? What if they target their ads better to reduce cost, but they're still sending ads? That's why the money goes to the recipient. After all, when a spammer spams you, its your time and aggravation they've wasted. Receiving an ad from a spammer who paid *someone else* a penny is no less annoying to the recipient. But receiving an ad with a penny on it creates balance. A payment of five cents is even better. If I had a nickel for every spam I got, it would pay for my internet connection. Each person should be free to decide their own threshold, the price at which they will not be irritated to receive an email ad.

Of course, you and I do not want to pay a penny or a nickel to send emails. We shouldn't have to. The only time email should cost money is when it is sent to a stranger, and only then if the message is unwelcome. This can be accomplished by attaching a small bond the first time an email is sent to a new contact. If the recipient cashes the bond, the cost is only a few cents and the sender probably won't email that person again. If the recipient does not cash the bond, there is no cost, and from then on the new contacts can communicate freely.

A sender should always choose a bond amount that they would be willing to lose, since there is a risk involved. Within this restriction, the sender should choose the largest possible bond. If the sender chooses a bond value that is smaller than their maximum comfortable value, then the message may be bounced back and have to be resent. If the sender risks the most that they are willing to risk, then there is no need to ever increase the bond and resend a message.

If a message is secured with a bond that is smaller than the recipient's threshold, the message should be bounced back rather than discarded. Risking any amount of money should at least entitle the sender to a notification if the message is not delivered.

1.1.3 Should email be signed?

Digital signing is a cryptographic operation that combines the contents of a message with a unique secret known only to the sender to create a "digital signature" for the message. Because the secret, called a private key, is required to create a signature, the signature cannot be forged. The content of the message cannot be changed after it is sent, or the signature will reveal the tampering.

Signature schemes have faced a few obstacles to deployment. Historically, the difficulty of setting up the necessary software and obtaining a signing key has been too much for most email users[37]. The software has been steadily improving to the point where these tasks should be within reach for anyone comfortable with the normal software installation process. Microsoft Outlook and Mozilla Thunderbird now both include signature support.

Widespread use of signatures also raises the question of who should be trusted to hand out keys and associate keys with particular email addresses. Large companies such as Thawte and Verisign offer this service. Thawte is offering free personal service to individuals; both companies charge businesses.

As using signatures gets easier, the primary obstacle has become simple lack of interest. I argue that the combination of spam, phishing, and identity theft plaguing us today make this a good time to start pushing signatures.

Signatures should be used on every email, every time as soon as possible. Email digital signatures prevent impersonation, a flaw in unsigned email that allows spammers to sneak through filters by pretending to be the recipient's best friend or grandmother. Using strict content evaluation filtering on all incoming email would have a high false positive rate, so messages from known senders should be exempted from the filtering. This opens the impersonation loophole that lets the spammers in. Using signatures closes this loophole.

Many people wonder how spammers know who to pretend to be, and there are a few ways. Viruses infecting users' machines can harvest address books, which give a

"circle of friends." Sending "from" one of these addresses to another, the spammer is likely to exploit an existing relationship. Another method is to record proximity of addresses when they are harvested off web sites. In the end, it doesn't matter how they are doing it. If impersonation is the best loophole open to them, they will find a way to exploit it. Signatures make this part of their jobs *much* harder.

Viruses infecting personal computers may be able to send spam signed with the key stored on the computer. This is unfortunate, but it is still better than the situation with viruses today. When a virus is sent out from an infected computer, it is often sent "from" a forged address. Well-meaning recipients reply to that address with a warning about the virus, but the person receiving the warnings isn't the person with the actual virus infection. However, if a virus wants to use the signature key on the infected computer, it will have to be honest about whose account it is using. Friends of that person will receive the virus' emails, but they will know exactly whose computer sent them and the infection can be tracked down and cleaned immediately.

1.1.4 Who should control infrastructure?

Systems exist today[33] that allow all users of the system to send each other email and vote about who is sending spam, and punish those people, and so forth. As closed systems, they may succeed in controlling communication within their communities. The crucial problem with closed systems is that their usefulness is proportional to their market share. When they interoperate with the outside world, they give no benefit above normal email.

If the product is truly excellent, it may be widely adopted and could become quite useful. This puts the central control over who can email whom into the hands of one organization. This organization then has the power to dictate what types of email are acceptable, creating a risk of censorship by management with a political agenda or profit motive. The organization could decide to sell advertising rights to marketers, and deliver the ads directly to their users.

If users are unwilling to abandon the system, they are open to abuse. If they are willing to leave, the system becomes gradually less useful and eventually fails.

A more fair and stable solution is for many groups to interoperate, so that users can choose who to do or not to do business with. An open, non-proprietary protocol will allow gradual deployment and free-market decentralized control. We already have fully open PGP signatures, and semi-proprietary S/MIME signatures. Email itself is a perfect example of a decentralized, interoperative system. The last missing building block is a non-proprietary financial transaction protocol.

Apuma creates a framework for such a protocol to be plugged directly into the filter, in concert with automatic accept-listing.

1.2 Overview of this document

The remainder of this thesis describes the implementation of Apuma, a modular spam filter employing an automanaged accept-list and configurable plugins, and elaborates on the idea that such a filter is usable and effective, and is more conducive to the healthy survival of email than any of the existing anti-spam solutions.

The primary contributions of this thesis are the implementation described in Chapters 3, 5, and 6, the filtering rulesets described in Chapter 4, and the ideas discussed in Chapters 1, 2, and 7.

Chapter 2 presents an overview of the categories of existing spam solutions. Each one is analyzed for strengths and weaknesses.

Chapter 3 analyzes the types of target users of Apuma, and their needs, then describes the user experience of working with Apuma.

Chapter 4 dissects the default filtering rulesets for both standard usage and parental usage.

Chapter 5 describes the implementation of Apuma, and how to write plugins for it.

Chapter 6 reports the results of usability testing performed on the Apuma GUI.

Chapter 7 proposes a few different ways of creating a financial transaction protocol, and discusses the issues that such a protocol will have to overcome.

Chapter 8 summarizes the ideas and contributions of this thesis, and future directions for this research.

Chapter 2

Related Work

The existing efforts in the spam fighting domain fall into roughly four categories: legislation, recipient filtering, sender payment and accept-listing.

2.1 Legislation

Laws such as the CAN-SPAM Act[20] make certain types of spamming illegal. They focus on mandating certain good practices such as giving a real return address and a descriptive subject line, and honoring opt-out requests. In many cases spammers are already committing other crimes such as fraud, and could be prosecuted under more general laws. Specific anti-spam legislation can help to prosecute spammers when other crimes are not occurring or cannot be proved. However, the legislative solution has problems with jurisdiction, since the internet is a global environment. Also, a spammer must be identified and found before they can be subject to legal action.

Some efforts are being made to use existing laws, such as copyright, in novel ways. The Habeas company[3] gives known legitimate senders the right to use a certain haiku in the headers of their emails, which will get the message past a large number of participating spam filters. Spammers using the haiku without permission are in violation of copyright and are subject to legal action on those grounds even if they

cannot be proved to be spammers.

Criminal cases and civil suits are enjoying some success against the largest spammers [15][31], but the laws don't put any real power into the hands of the spam recipients. Anyone who receives an illegal spam cannot do much beyond forwarding the message to `spam@uce.gov` and hoping that the spammer will soon be identified and punished.

2.2 Recipient filtering

Many filtering solutions exist that evaluate a message based on some set of criteria such as content and headers and decide if it is spam. These filters can either be applied at the ISP level or at the client level. If done by an ISP, care should be taken to ensure users have a chance to list exempt senders; otherwise users may suffer falsely rejected mail and cannot remedy the situation. Although some ISPs offer this, many users are not aware of the option and do not take advantage of it. Some ISPs do not give users any control over their mail filtering.

2.2.1 Keyword and rule based filters

Rule systems are developed based on existing spam content, usually giving a score to each keyword or message characteristic. The scores are totaled and compared against a threshold value to decide if the message is spam. Choosing a threshold value is the difficult part of using this type of filter. Without accept-listing, a low threshold will cause false positives. A high threshold will allow too much missed spam.

The most popular system available today is SpamAssassin [9]. SpamAssassin is free for use by ISPs, businesses, and individuals. Other companies simplify the process of using SpamAssassin by packaging it as a service or an appliance, and selling it to corporations as a turn-key anti-spam solution[2].

The writer of a rule set has control over what is deemed spam, but users may

be able to read and edit the rules. Spammers will constantly evolve new tricks to circumvent the existing rules, so someone must perpetually update the rules to match the latest spam. The workload on the rule-list maintainer can be lightened if other users design and submit rule patterns for new types of spam. There is no change to the email infrastructure required since every user's filter operates in isolation.

Spam that uses the latest techniques will slip past these filters. If the text contents of the spam are rendered as an image, then keyword filters will have no content information to process. This is one way the spammers try to hide from such filters.

Keyword filters are prone to flag legitimate mail that happens to discuss common spam topics. Legitimate opt-in mailing lists are also at risk. Newsletter writers have had to become careful about what words and phrases they use, or their subscribers may not get their messages[32].

2.2.2 Machine learning

Many algorithms exist which attempt to identify spam with rules learned automatically from a corpus of pre-classified email. The user can continue to fine-tune the training simply by flagging messages as spam or not spam as they arrive. This allows the filter to learn the individual user's email patterns, and hopefully become more accurate on that user's email. Spam that looks nothing like previous spam, or that looks something like previous non-spam, will get through. Malicious spammers can include lines or passages of normal text along with the spam, so that if the user flags these messages, the filter may become confused about words and phrases that appear in the text. For example, a learning filter may start to block all messages with the subject "Meeting rescheduled" which would eliminate a great number of spams but could have dire consequences. A large variety of training algorithms exist, and some are better than others at not making these kinds of mistakes, but none are perfect. Some algorithms are available for use with existing mail clients[21], others are explored only in theory[22]. Many newer mail clients have a trainable junk-mail filter

built-in[12].

2.2.3 Collaborative real-time spam characterization

A large number of users working together can flag spam messages they receive, and build up a set of "message signatures" which allow similar spams to be blocked in real-time. The advantage of this over a handmade and distributed rule set is that it can respond much more quickly to the current incarnation of spam.

Vipul's Razor[13] is one such system. The credibility of a spam report is derived from the user's reputation, based on past reports. A distributed, collaborative, reputation based system relies on the community property that the majority of users are honest. If a community can be overwhelmed by bots run by spammers, then the consensus of the group cannot be trusted. Vipul's Razor is in the process of implementing a challenge-response feature in its registration process, but it currently seems vulnerable to this kind of attack.

2.2.4 IP block-listing

The IP addresses used by known spammers can be compiled into a list against which incoming messages' headers are compared[11]. The maintainers and distributors of such a list have the power to deliberately shut off email for an innocent party[27], but the existence of multiple, parallel lists reduces the ability for abuse. The maintainer of the list must make decisions about whether to block the mail server of an ISP if some of their customers are spamming and some are not. Innocent customers of ISPs that do business with spammers can lose the ability to send mail[24]. Sometimes IP segments approaching the range of whole countries are blocked out[17]. Based on user submitted spam samples, the list maintainers have the final say about who is a spammer.

Making these lists and maintaining them requires a great deal of human effort. For efficiency, a small number of lists should be shared by a wide user base. Methods

for distributing this list are needed, as well as methods for reporting spammers and arbitrating complaints from those who feel they have been unfairly added to the list. Spam sent from previously unknown spamming hosts, such as trojan infected home users, will not be stopped by this kind of filtering for the time it takes the maintainers to register the complaints and list to be updated and disseminated. This means that the list updates must be very responsive, and some spam will always get through.

2.3 Sender payment

Sender payment methods allow the sender to demonstrate eagerness to deliver a message by risking money or expending effort. While this cannot directly stop spam, it aims to increase the cost so that the economics of spamming are no longer profitable. The recipient has the option to either reject all messages that do not demonstrate eagerness, or accept all the ones that do and filter the rest by a secondary method. Rejecting all non-participating messages would not be feasible until a particular solution had developed a significant market share. All sender payment schemes are designed to have economic impact on the sender, and therefore they put the desirable free qualities of email at risk. Their use should be considered carefully lest email become a for-pay broadcast advertising medium like television and radio.

All sender payment systems must also face the problem of mailing lists. Mailing lists either require separate sender effort for each subscriber, making it too onerous a task to send to a large list, or they allow a sender to leverage a single payment to a large number of recipients. In general, mailing list policies would need to be determined on a case by case basis by the mailing list maintainer. Many mailing lists and discussion groups are already closed or moderated for other reasons. If a mailing list does not have a policy of who can post messages, or what kind of messages can be posted, then anyone has a right to post whatever they want. Users who subscribe to public, unmoderated mailing lists are, in essence, inviting the world

to send them mail. Removing all spam from such lists is one of the hardest problems in the spam domain. As I have argued previously, automated filtering can only work well in conjunction with a well maintained accept-list, and this is as true for lists as for individual mail.

Sender payment systems fall into four categories: proof of computational effort allows the senders to pay with computer time rather than money, large bonds allow companies to put thousands of dollars in an account which is forfeited if they send spam, micro-bonds allow individuals to secure single messages with a tiny amount of risked money, and micro-payments cause the sender to give away a small amount of money for a sent message whether it is spam or not. All these methods guarantee that a message will not be stopped by a spam filter that recognizes the payment.

2.3.1 Proof of computational effort

Proof-of-work proposals require every email to include a token, proving that a certain amount of computational effort was performed by the sender. Because the computation must be performed for every message, and each computation takes time, a sender's email rate is throttled to a level decided by the recipient who approves the token. It is important to set the threshold just right since spammers can still send spam at a limited rate. Spammers' control over large numbers of zombie hosts make this difficult, since the threshold must still be high enough to allow legitimate mail to proceed comfortably[26]. Preliminary calculations suggest that the availability of cheap computers and zombie hosts may mean there is no acceptable threshold to use if computation is required for every email. However, if combined with another system to exempt mail from known senders the requirement could be raised. Spam would be reduced by up to 350 times. according to HashCash[14], the primary supporter of the scheme. They predict it will be 10 years before the adoption of HashCash is complete enough to start bouncing messages sent without tokens.

2.3.2 Large bond

Big senders can put up huge amounts of money to "buy trust" from a company such as IronPort[4], who in turn will route their messages past the spam filters of participating companies. Since the bonds are on the order of tens of thousands of dollars, this is clearly a solution only for large corporate senders. Individual senders, and even small businesses, will not find much use in this. It is useful to ensure delivery of legitimate commercial email, but recipients can only use this as an extra accept-flag, so other filtering is still needed. Customers can complain if they get spammed, and the bond holder must be trusted to arbitrate complaints and decide when to stop verifying a sender. If a majority of commercial mail is bonded, then non-participating commercial senders are more likely to be treated with suspicion. There should not be any other negative impact since it identifies legitimate mail, not spam, so there cannot be false positives. The bond holders need to set up an infrastructure to provide a method of verifying that a message came from a bonded sender. With IronPort, this is done by listing the mail servers of bonded senders. If bonded senders fail to secure their mail servers, spammers may exploit them, causing the bonded sender to risk losing their bond money.

2.3.3 Micro bond

If both the sender and recipient of a message have accounts with a compatible financial infrastructure, the sender can risk a small sum of money on a message to ensure delivery. The recipient has the option to cash the bond if the message was unwanted. A financial third party is needed to hold the funds. Multiple financial institutions, such as existing banks, or ISPs, could potentially inter-operate, and systems could be built on top of existing financial infrastructure. The user can decide their own "cost of delivery," the payment level at which they are perfectly happy to receive spam. If too much spam comes, they can raise the cost until they feel they are being compensated for the inconvenience of deleting it. Bonded messages below this threshold should be

bounced back to the sender with "insufficient postage," unless the sender has indicated they don't want bounces. Senders could indicate in the bond if they wanted to see bounces, as personal senders would, or let the message drop as an advertiser might. Messages with no bond could be bounced with an informative message so the sender can learn about bonding. The bond required on a message could be dynamically calculated from the message's content evaluation score, to charge suspicious looking messages more.

Marketers may be willing to risk a few cents to get through, so not all advertising would be stopped. This is arguably a benefit of this system, since it takes into account the interests of legitimate marketers who want to reach potential customers with ads that are not fraud or scams.

Because the sender and recipient must both have compatible accounts, either multiple systems must inter-operate or a single system must have total market control. An example of a self-contained proprietary implementation is Vanquish[33]. There are no open implementations in use at this time.

2.3.4 Micro payment

Micro payments are similar to micro bonds, except the money is paid up front rather than at the recipient's request. A friendly implementation would allow the recipient to send the money back. However, payments have the advantage, compared to bonds, that they can easily be sent anonymously since no account is required on the sender's end. A one-time payment could be purchased from the recipient's bank and included in a message, or as an attachment, with no special software. Once again, marketers can spend a few cents to get through to customers. A few companies such as PepperCoin[34] currently offer micro payments, but none of them are working in the email domain.

2.4 Accept-listing

Filtering spam becomes easier if known legitimate senders are put on an accept-list and the remaining mail can be treated with higher suspicion or outright rejected. The list can be generated by hand, or automatically from all previous correspondences. Once the list is in place, there are several methods to prevent a spammer from impersonating someone on the list.

2.4.1 IP accept-listing

IP accept-listing is the opposite of IP block-listing; a mail server not on the list is not allowed to send mail, or is restricted in mail volume. Senders must petition the list maintainer to be added to the list. If multiple competing lists exist, then a maintainer of a mailserver must track down and petition every accept-list being used by anyone they may want to send email to. This is an unscalable problem unless either a single list is centrally maintained, or several lists are maintained and recipients filter using the union of multiple lists. When an ISP maintains an accept-list for private use in-house, the customers of that ISP may not receive mail from senders that have not petitioned the ISP for access. Only the very biggest recipients, such as AOL, can afford to demand participation from all senders. AOL requires bulk senders, such as newsletters, to fill out an online "whitelist request form."[1]

Accept-listed hosts that fall victim to trojans or hacking can send spam until they are removed from the list. Unestablished senders are locked out of the system until they can prove themselves somehow. Most legitimate mail is concentrated through ISP mail servers, so even if the list is only required for bulk mailing, nearly every message will be affected. Every single ISP will need to negotiate their way onto the list, which is troublesome for the mailserver administrators.

2.4.2 Individual keys used to sign messages

Digital signatures can be used to allow incoming mail from accept-listed senders, without allowing impersonation. Each sender generates and safeguards a secret key. Recipients maintain a store of known senders' public keys, which allow them to verify the identity of anyone they have previously dealt with. Keys cannot automatically be trusted to establish identity, unless they are verified personally, but they can be trusted to represent the same entity each time. Trusted parties can sign keys to give them more authenticity as a form of ID. Certificate authorities can issue certificates associating a key with a certain email address.

Trojan infected machines can send spam signed with their owners' keys, but since the "From:" address cannot be forged, the infection is very easily traced and repaired. If a sender with an established key loses or changes their key, that would appear like an impersonation attempt, and some mail may be rejected for a while. PGP[10], GPG and S/MIME[23] are common implementations of this idea.

2.4.3 Domain based authentication

Domain based authentication works by allowing the administrator of a sending domain to declare what messages are truly "from" that domain. Senders must prove a relationship with the domain they claim to be from, by sending messages through the domain's authorized server. If the server is properly secured, this will prevent outsiders faking affiliation with a domain. Recipients can verify the domain of the sender, and reject unproved messages from participating domains. Since all legitimate mail from a participating domain will prove itself, it is safe to outright reject other messages from these domains. The mail servers must be trusted to authenticate users properly, and the domain registrars must be trusted to decide who controls a domain. Mail from non-participating domains should not be affected, but some recipients may enact policies requiring participation, or heavily filter mail from unproved domains. Internet wide infrastructure changes are required, but they are minor. Small

changes to DNS enable the domain record to carry the needed information. Domain based authentication can be rolled out one sending domain at a time, as long as non-participating domains are tolerated. Existing systems include SPF[7], which is an open standard, Yahoo's DomainKeys[38], and Microsoft's SenderID[29]. SenderID is very similar to SPF, but it is a proprietary system and yields no advantage over the open standard. SPF and DomainKeys do not interfere with one another, so there is no pressure for mail server administrators to guess which one is better. Both systems can be used simultaneously by a domain for outgoing messages, and a filter system can choose to honor either or both.

Domain based authentication cannot prevent all impersonation. Since only the domain is proved, not the user name, one user of a domain could potentially impersonate another depending on server configuration. As always, zombie hosts can send "From:" their owners. Senders who use their ISP's domain but their own mail server look like forgers. However this situation is rare since domains are very cheap.

2.4.4 Turing test challenge-response

If a non-spammer finds that they are being bounced because they are not on an accept-list, they need a way to get onto the list. One such way is for the email bounce to include a Reverse Turing Test (RTT), which gives the user a chance to prove he is a live person and not a spamming robot[36][30]. After completing the challenge he is added to the list and not challenged again. Recipients can set up software to generate challenges from their computer, or subscribe to a service such as the popular MailBlocks[5] to handle it all for them. Some systems allow a sender to prove themselves once to the service as a whole, and not to every single recipient, which saves some trouble. Dealing with the challenges is still an annoyance to legitimate senders, who often resent the recipient for saving themselves the hassle of spam at the expense of others. Waiting for the extra steps of a challenge response turnaround can delay an email for hours or days and disrupts the asynchronous nature of email. In

some cases, emails have been lost forever because the challenge message was filtered as spam[16]. Corporate senders such as Amazon must make arrangements with the challenge response service, or require users to manually update their accept-list.

The graphic-only implementations of RTTs are not compatible with screen readers, which discriminates against the blind, but this is fixed by offering an alternate audio version of the test [35]. MailBlocks offers only a graphical test, and blind users must contact customer support for assistance. The widespread need to identify real human users has led to RTT accessibility problems in many contexts. HotMail and PayPal offer audio accessible versions of their RTT during account creation, but Yahoo Mail and Lycos Mail currently do not.

The only reason these tests are a reasonable way to manage an accept-list is that spam is assumed to have fake from addresses, or no live people watching the bounces. However, it is not impossible for spammers to manage to meet the challenge, by a sophisticated automated system or a massive amount of cheap/free labor. If answering a single challenge gets a spammer access to all the users of a certain service, it is well worth his while find a way to dodge the RTTs.

2.5 Hiding from spam

Whether or not an email user chooses to filter their incoming email, they can also try to avoid getting spammed in the first place. It is generally a good idea to avoid revealing an email address on websites, chat rooms, or newsgroups, where address harvesters are likely to pick it up. When submitting personal information to businesses, it is a good idea to carefully read the privacy policy that governs how that personal information will be handled, and opt-out of any promotional mailings. The Federal Trade Commission maintains a list of advice for consumers[19].

Some novel address-hiding solutions exist, such as giving unique, disposable, computer-generated email addresses to every potential correspondent. SpamMotel[8]

is one such solution that gives users a desktop application that generates addresses for web forms. Users can also generate addresses through the SpamMotel website. If a certain incoming address starts receiving spam, the user can block off that address or associate it with a certain sender. Thus they can continue to receive mail from the sender they created the address for, but not get spam. This solution does not allow the user to have a single, publically known, contact address, and the generated addresses are not memorable or human readable. The solution is therefore somewhat awkward for interpersonal communications. It may be very helpful to users who are worried about engaging in e-commerce.

Chapter 3

User Interface

Apuma's behavior can be customized by modifying the configuration file or installing new plugins. This chapter describes the user's experience when using the default rule set.

3.1 User analysis

The User Interface was designed to meet the needs of certain groups of users, whose needs are discussed below.

3.1.1 The user's concern

Users want to see the correspondences from people they know, and organizations they work with, without having to sift them from a heap of ads and scams. Users can delete the spam, but the more junk there is, there more likely they are to accidentally trash a real message.

Users want to read their mail without spending time dealing with spam, and they want to be sure that none of their real mail is getting lost. In order to set up a filter that will reject things that look like spam, there has to be a way to make absolutely sure that non-spam gets through.

3.1.2 Who are the users?

The following user groups are considered the targets for the usability of the system.

Novice and Casual Users (home email)

Home users of email are usually technically unsavvy, they don't know how things work and they often don't want to know. They want things to "just work" without effort or interaction. They probably check their email much less frequently than other user groups, and the subsequent build-up of spam is very burdensome. (Because their legitimate email is low volume, the percent of spam also is much higher.) To many, the internet is confusing and scary; they do not trust their understanding of it and they do not trust themselves to make decisions about it. They don't want to be given a chance to make mistakes they can't undo. They know they are responsible for looking out for themselves online, but they don't know how. The sensationalized news stories about virus epidemics and identity theft have left them suspicious and exhausted; they are becoming impatient with the shortcomings of the internet. Inexperienced email users tend strongly to be older and either retired or working in nontechnical fields. They are almost certainly unfamiliar with technical terminology of email, networking, security and encryption, and some may not speak English at all.

Experienced and Expert Users (home and work email)

Experienced users are likely to rely on email for important personal and work communication. In many cases a single lost message will be unacceptable. They have a higher level technical understanding of how email works, but they do not want to spend time worrying about maintaining their email setup. Email is an established part of their daily routine, and they are comfortable with it. Many check their email every few minutes, so checking email should take as little time and effort as possible. They may use several email accounts, and email clients, for separating home and work roles. Experienced users may not be English speakers, but they will have mastered

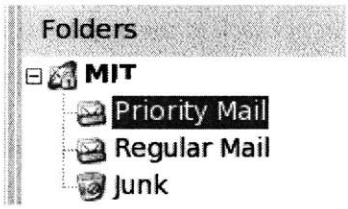

Figure 3-1: Folder view

the terminology of the domain.

Parents

Parents may use a filtering system to protect their children from inappropriate email. Parents will have a wide range of email experience, but they may be total novices. Because they are seeking protection, they will prefer to err on the side of stringency when filtering messages. The cost of undesired emails reaching their children is probably viewed as much higher than the cost of their children's email being delayed or lost.

3.2 Message annotation

The starting point for the user's interaction is their ordinary mail client, where they will see email messages that have been filtered, sorted, and annotated by Apuma.

The user will see that their new mail is filtered into three folders: "Priority Mail," "Regular Mail," and "Junk Mail." See Figure 3-1.

The "Priority Mail" folder is for messages that are almost definitely good. This mail comes from senders on the accept-list, and should be signed by a personal sender key. It will also be accepted if the sender is not known to have a key and the domain information does not appear forged.

"Regular Mail" is available for unapproved senders. A sender not on the approved list *cannot* get a message into the Priority inbox. They can guarantee delivery into "Regular Mail" by attaching a sufficient micro-bond.

"Junk Mail" will get some or all of the messages that didn't make it into the other folders. The user can set a cutoff point between 0-100% on the numerical output of the content evaluation filter. Messages below the threshold are delivered to "Regular Mail." Messages above the threshold are delivered to the third folder, "Junk Mail." Messages in the Junk folder have their spam score prepended to their subject line. This makes it easy in any email client to sort messages by score, thus displaying the least spammy messages first.

These folders were renamed from "Inbox 1" and "Inbox 2" based on users' remarks.

When the user views a message, there is a bar of small text indicating a variety of potential message traits: known sender, signed, encrypted, bonded, or spam. Then there is a [More Info and Options] link. The text and link are inside a thin colored box that gives an intuitive overview of the text. Trusted senders are flagged with green, spam with red, and uncertain emails with yellow. The information expressed by the color is fully duplicated in the text for color blind users.

Clicking the link goes to the message specific info page. This page explains the meaning of each term (digitally signed, etc) and why it was or was not applied to that message.

Possible actions are afforded by big buttons. UI evaluation suggested that this would be better than links within the text. The buttons are easier to find because they don't blend into the text, and the graphics allow more visual distinction between different actions. See Figure 3-2.

3.3 Manual contact list management

Editing the Known Senders list is done with an interactive table. Originally, and through the computer prototype phase, this was planned to be an html form with the text boxes contained in an html table. Not only did this look awful, it was impossible to add incremental search. So I switched to java for the final implementation, and

Figure 3-2: A typical Apuma GUI intro page

the look and feel are much better.

The list editing box displays the contact list on the left and the details of the currently selected contact on the right. This was done because the user fields stack better vertically. The horizontal space was already getting tight with just a few fields, and that design does not allow for any new fields to be added later. The table display on the left is limited to just the data needed to identify a particular user: name, email, and possibly nickname.

When other parts of the interface reference a contact, the nickname is used if one is available. If not, the name is used. If no name is set, the email address is displayed. The email address is the last resort because it is probably the least recognizable string associated with a contact.

An incremental search feature allows the user to enter any text into the search box and all the contacts' names, emails, and nicknames will be searched as the user types, updating the table in real time. See Figure 3-3.

The method of deleting contacts was a tricky question. One solution which benefits from external consistency is to make deleted contacts disappear and let the user restore them in reverse order with an undo command. I felt this was unnecessarily

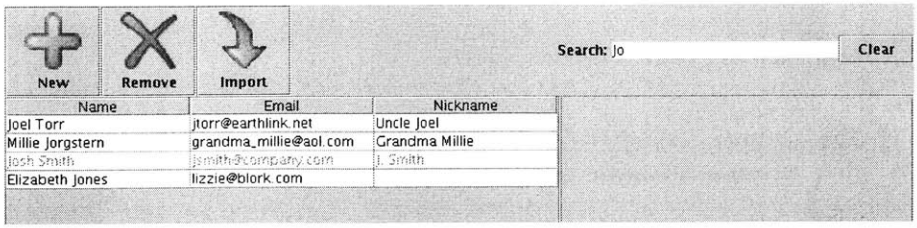

Figure 3-3: Incremental search

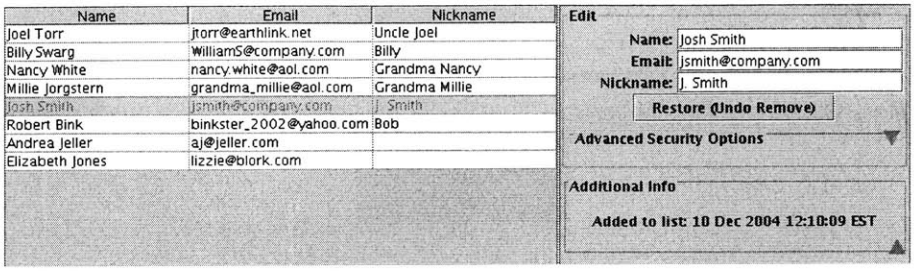

Figure 3-4: Deleting a contact

restrictive because the operations are independent so there is no real reason they have to be undone in reverse order. See Figure 3-4.

Since the "flag deleted" and "clear deleted flag" model creates surprise in user tests, I have determined that it is not the best way, although it is best among the simple to implement options. Probably the best way is to use the TrashCan model, since deleting files is a very similar operation which should be undoable in any order.

3.4 Automatic contact list management

The automatic management of the accept-list is meant to allow complete hands-off operation in almost all cases. This functionality attempts to capture the social network of the user by monitoring incoming and outgoing mail.

The addressees of outgoing mail are presumed to always be desired contacts. All To and CC addresses on an outgoing message are added.

Incoming messages from known senders may have multiple To or CC addresses, which are all added to the accept-list. This captures the notion of an "introduction,"

allowing two people with a common contact to communicate with each other. To prevent exploitability, this rule only applies if the original message was digitally signed by the existing contact.

A further idea (not implemented) would be to capture any addresses in the body of emails as well.

Chapter 4

Rulesets

4.1 Message processing

The steps for processing a message are determined by a special configuration file that determines which modules and plugins are used. This file can be edited by the user to create custom configurations. Two default configurations are described below.

4.1.1 Standard configuration

The standard configuration sorts mail into three categories. Priority Mail is from known senders. Regular Mail is guaranteed with a micro payment or bond, or is highly unlikely to be spam based on content evaluation. Junk Mail is mail from unknown senders that looks suspiciously like spam.

Figure 4-1 shows a flowchart of the processing steps.

The first rule checks whether the message is signed. If the message is signed, then the accept list is checked for the sender's name. If the sender is a known contact and the message is signed, then it is delivered immediately to Priority Mail.

Unsigned messages may have forged sender information. To protect against this, unsigned messages are checked against any available domain verification technologies such as SPF or DomainKeys. If the result indicates that the message is forged, it is

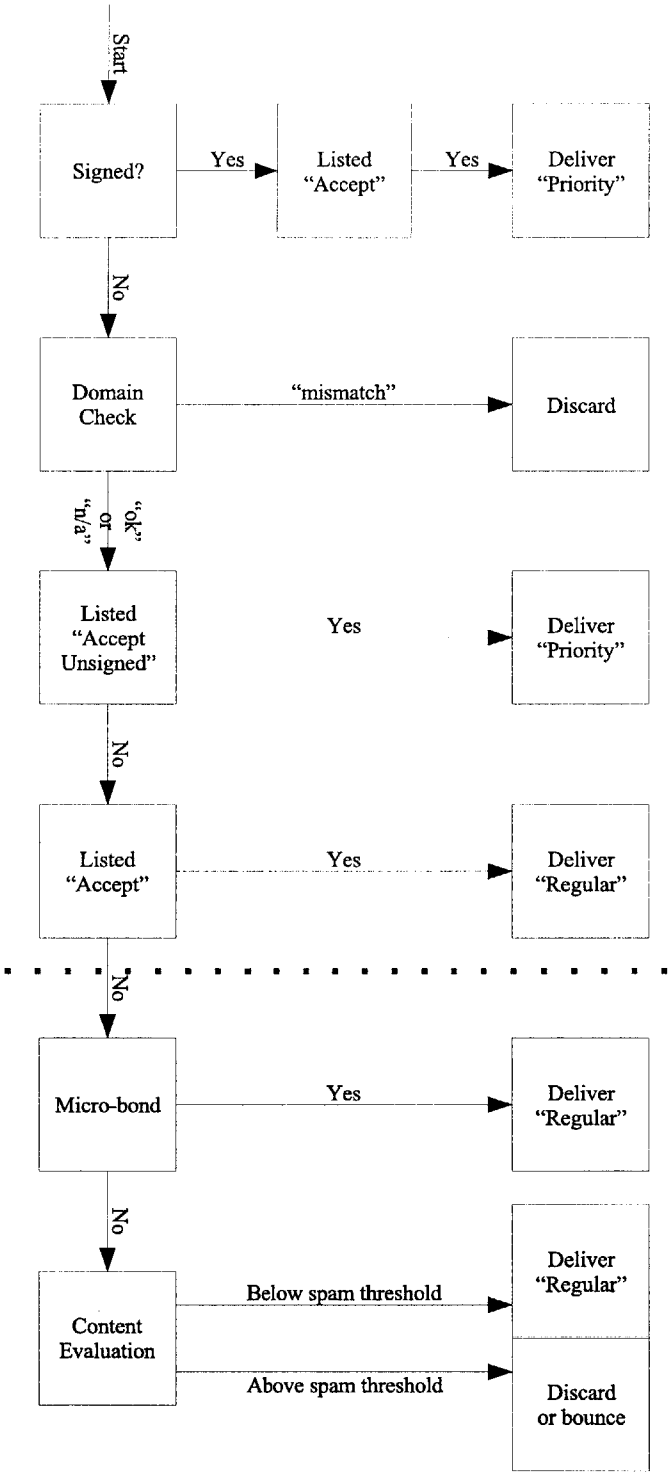

Figure 4-1: The standard configuration ruleset

rejected. This rule can only reject mail claiming to be from domains that have chosen to participate in an anti-forgery technology. A bad result here means the message is highly likely to be a forgery. If the test is acceptable or inconclusive, the message continues to the next stage of processing.

Unsigned messages are checked against the contact list. If the sender is on the accept list, and has never sent signed messages, or is flagged for "accept unsigned messages", the message is delivered to Priority Mail. If the sender is on the accept list but has in the past sent signed mail, the message is delivered to Regular Mail. If the sender is not on the list, the message continues to the next stage of processing.

If the message is guaranteed with a micro-payment or micro-bond that is at or above the user's threshold, it is delivered to Regular Mail. Choosing a threshold is discussed later in Section 7.4.

At this point any remaining message is from an unknown sender, and has no payment or bond to guarantee it. This is the class of messages that is treated with the highest suspicion. If content evaluation indicates that is it almost definitely not spam, it will be delivered to Regular Mail. If content evaluation suggests that it is probably spam, it is delivered to Junk Mail with an annotated subject line that allows the Junk folder to be sorted by spam likelihood. If content evaluation indicates that it is almost definitely spam, it can be simply discarded.

4.1.2 Parental control configuration

Parents of children who use email have a strong need to prevent inappropriate material from reaching their children's eyes. The Apuma ruleset for children's email strives to ensure 100% filtering of unwanted email. Only email from known senders is allowed in. The standard ruleset proceeds up to the dotted line in Figure 4-1. Any message crossing below the dotted line is considered questionable, and is re-directed to a parent's box. The parent can review the message and add the sender to the accept-list if desired.

The goal of the child-safe filtering is *not* to prevent the child from deliberately contacting anyone, or accessing any material they choose. (An enterprising child can shut off or circumvent any filter that gets in their way.) The goal is simply to shield the child from the incoming offensive and inappropriate material that is randomly targeted to all inboxes. Therefore, the accept-list is managed the same way as it is with an adult's system.

Chapter 5

Implementation

The driving factors in the design of Apuma were ease of use, extensibility, non exploitability, platform independence, and mail client independence. The system is intended to work equally well whether it is installed by the user or the ISP.

5.1 System overview

The basic structure of Apuma can be seen in Figure 5-1. The diagram shows the components as if the system were deployed by an ISP. In that case the user's computer would not have any new software installed on it. If, on the other hand, the user were to install Apuma themselves, then all the components shown in the box "ISP's Server" would run locally on the user's computer. All the functionality would remain the same.

Incoming messages (shown by the downward arrows) are passed through the filter system. This Apuma component uses processing plugins to analyze and annotate the message. Plugins will be described in detail later on. After processing, the messages are read in the user's normal email client.

Outgoing messages (upward arrows) are composed in the normal client, and then sent through a proxy that adds a digital signature and updates the accept-list to

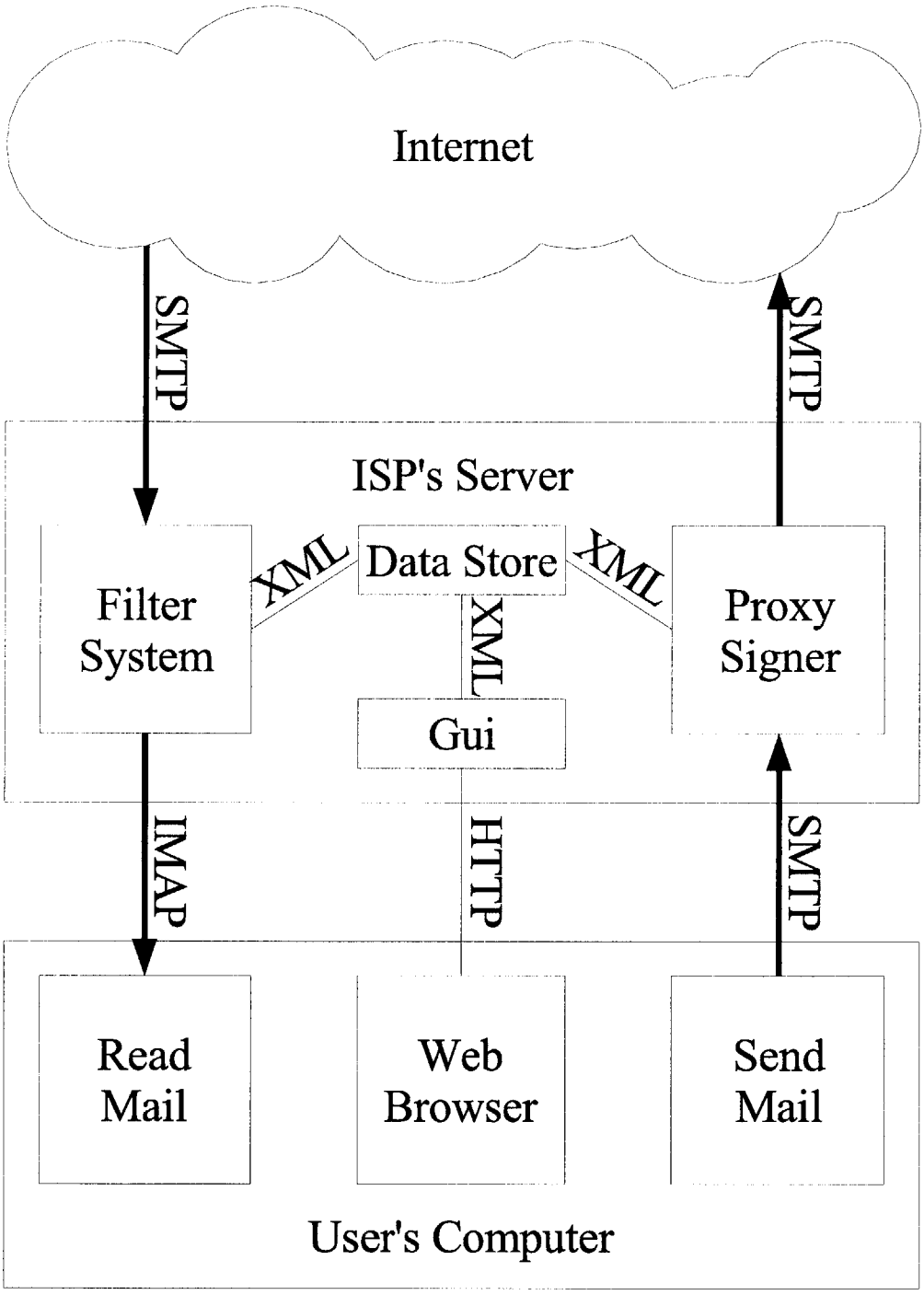

Figure 5-1: The Apuma system structure

include recipients of the message.

The graphical interface is a java applet served up by a special web server (labeled "Gui" in the diagram.) The applets are viewed locally in the user's web browser.

The filter, the GUI, and the outgoing proxy all rely on an XML data store that houses the information about contacts and the user's configuration files.

5.2 Contact list data

Information about contacts is stored in XML files, one per contact. The files are named by the email address of the contact. They are all stored in a subdirectory of the user's .apuma configuration directory. This format allows a measure of transparency, if the user is curious to explore the list by hand. The files themselves are human readable, and individual items can be found and selected with the user's preferred file browser. The more important reason for this format is scalability. When processing a message, the filter only needs to load a single contact file off the disk. If all the contacts were stored in a single file, the speed of processing each message would be affected by the total list size.

Each contact file stores the email address (in case it contains special characters that had to be stripped out to form a legal filename), and a data section that is subdivided by which plugin owns the data. The special plugin section "Info" stores the full name, nickname, and information about when and why the contact was added.

The contact list should maintain itself during the use of the system (as discussed in section 3.4,) but to get started the contact list can be generated automatically by the script populate.py. This script populates the contact list by connecting to an IMAP mailserver and looking at the senders of all stored messages. Folders with names including "junk", "spam", "bulk", "trash", "archive", or "backup" are skipped. Although users differ, some people archive all their mail, including spam. It is assumed that after these folders are ignored, any remaining mail represents a desired

communication, and thus the sender's details should be recorded.

The following is an example of a typical contact file, created by populate.py.

```
<Correspondent>
 <Data>
  <Info>
   <Address>simsong@lcs.mit.edu</Address>
   <Added>Added 2005-05-01 17:11:17.677673 because of stored message \
"Re: Meeting regarding antispam MEng thesis" from Tue, 13 Apr 2004 \
06:13:42 -0400.</Added>
   <Name>Simson L. Garfinkel</Name>
  </Info>
 <Listed>
  <Accept>True</Accept>
 </Listed>
 </Data>
</Correspondent>
```

5.3 Filter

The filter is written in python for platform independence. The main filter class takes the text of an email, pushes it through a tree of processing plugins defined by a configuration file, and yields a modified email as output.

5.3.1 Incorporating the filter into mail handling

The Apuma filter is suitable to run on the mail server as a filter in procmail. If the user cannot access the mail server's configuration, the filter could run on the client machine, wrapped in a script (designed but not yet implemented) that would monitor an IMAP server and pass new messages through the filter.

Messages processed by Apuma are given an X-Apuma-Folder header to indicate where they should be delivered to. The content of this header is configurable, and defaults to 1, 2, or 3. It is up to the procmail recipe or IMAP script to read this

header and place the message in the right folder. This allows the filter to be used with any mail system.

5.3.2 Plugins

Plugins are decision-making modules that can be incorporated into the Apuma framework. A plugin analyses a message object, optionally mutates the message, and returns a value. The return value is used to decide which plugin the message gets passed to next.

Default

Apuma includes several default processing plugins.

- Check if sender is on a list

  The `Listed` plugin checks whether a contact is a member of a certain list, usually "Accept." It does this by loading the contact's file off the disk and checking the property for the list name. If the property is true, the plugin returns true. Otherwise, if the property is false, or the property does not exist, or the contact does not have a file, the plugin returns false.

- Check message signature

  The `S/Mime` plugin checks whether the message is signed. If not, it returns false. If the message is signed, and the signing key matches the key on record for the sender, it returns true. If the sender is known, but they have never before sent a signed message, or the sender is unknown, the key is recorded.

  This implementation of the `S/Mime` plugin depends on openssl.

- Create a link to access the Gui

  The `MakeLink` plugin generates a unique, random message identification code, then records information about the message in a file identified with the code.

The plugin adds an X-Apuma-Link header to the message. This link will load the Gui applets in the user's web browser.

- Modify message to display information

  The `UI` plugin adds a text bar and color cues to the top of the message. It also adds the link from the X-Apuma-Link header in to the message body. This plugin can be removed from the processing chain if a more invisible installation is desired.

- Check SpamAssassin score, and update subject line

  The `ContentEvaluation` plugin checks for an X-Spam-Score header, previously inserted by SpamAssassin, and returns its value. This plugin is also responsible for tagging the subject lines of spam messages so they can be sorted by score.

- Check SPF or DomainKeys records *Not yet implemented*

  The `DomainCheck` plugin will check whether the sender's domain can be verified with SPF or DomainKeys. It will return either "verified", "mismatch" or "n/a." A return value of "mismatch" is an important error condition that means a message's from address might have been forged.

- Forward to another address *Not yet implemented*

  The `Forward` plugin will forward a message to an address specified in the configuration file.

API

A plugin must be a python module that defines a class that extends `plugin`. If applicable based on its output value, the plugin class should extend `plugin_num` or `plugin_bool`. The class does its work in a method called `process`, which takes a message object (of the python email package) and returns a pair of a response value and modified message object. The response value can be None. It is fine to mutate

the original message, or create a new one. Within the process method, the plugin can do whatever it wants to the message.

The plugin can access the parameters that were applied to it in the configuration file. (See section 5.3.3.) The simple call `self.config(''paramname'')` will return the value of paramname, whether it was defined for the plugin class as a whole, or this specific instance. Specific instance parameters will take precedence in case of a conflict.

Plugins may import the module `debug`, which exposes `log(data)` and `error(data)`. In general, logging is silenced, but it is a good idea for plugins to mention bad occurrences to `error()` and unusual occurrences to `log()` to assist future debugging.

5.3.3 Configuration files

The configuration file defines the tree of plugins that a message will be passed through. The configuration file is XML, and uses an ApumaConfig tag to wrap all its content. The content is divided into GlobalConfig and ProcessTree. GlobalConfig defines what plugin modules are loaded, and sets parameters that apply across all instances of a plugin. Each Plugin tag also gives a name to the plugin, which will be used to reference it later. This extra level of indirection allows different plugins to be easily substituted into the same role.

```
<ApumaConfig>
 <GlobalConfig>
  <Plugin name="ContentEvaluation" module="plugin_ce" class="plugin_ce">
    <SpamHeaderName>X-Spam-Score</SpamHeaderName>
    <AlterSubject>false</AlterSubject>
  </Plugin>
...
```

The ProcessTree defines a set of Processor objects. Each one has a plugin associated with it, which defines what class it is an instance of. The Processor is also given an "id" which will be used to route messages to it. It then has an optional Config section that defines any parameters specific to that instance. Finally, it has a set of

zero or more Output tags. After the plugin finishes processing a message, its Output tags are considered in order. A processor can have as many outputs as desired. The first one that applies is used, and the message is routed on to the next plugin. If none are listed, or none apply, the processing stops and the message is delivered as it is.

For example, this processor uses ContentEvaluation to decide a spam score for the message. If the score is less than or equal to zero, it is passed to the plugin with id "yellow". Otherwise, it is passed to "ce2".

```
<Processor plugin="ContentEvaluation" id="ce1">
 <Config>
  <MarkHeader>X-APUMA-Spam</MarkHeader>
 </Config>
 <Output maxValue="0" next="yellow"/>
 <Output next="ce2"/>
</Processor>
```

Output tags have zero, one or two matching statements, and a next plugin. The possible matching statements for an output are "value", "maxValue", and"minValue". Outputs using a "value" statement require an exact match. A "value" statement must be the only statement in an output; "maxValue", and"minValue" can be used singly or together, to define a bounded range. A "value" statement will match a python string of the exact same content. If the return value is a number, the "value" parameter is converted to a number and compared. If the return value is a boolean, True will case-insensitively match "true", "yes", or "1". False will match "false", "no", or "0".

There is nothing to stop a message from passing through the same Processor more than once; this is perfectly valid because if the message is being mutated there will not necessarily be an infinite loop. However, a config file can be written that will loop forever. The ProcessTree structure is intended to be very flexible and powerful, and thus has no arbitrary restrictions on processing complexity. To assist debugging, the filter will print a warning if it passes a message through 100 processing steps.

5.4 Proxy

The proxy is a python program that listens as an SMTP server and processes outgoing messages. The proxy first removes any Apuma links from messages being forwarded or replied to. Then it signs the outgoing message and sends it. To and CC addresses are added to the accept-list.

Signing messages relies on openssl. This functionality is adapted from a script written by Simson Garfinkel.

5.5 UI

The Apuma user interface is a set of java applets; java was chosen for platform independence. The applets launch from links inserted into emails. Each link has a message identification code in it, which is used to retrieve the data (sender, signed status, SpamAssassin score) associated with that message in the database. A python script functioning as a web server translates the message identification code into an html page with the data in a java applet's `param` tags. The data in the html file is then available to the java applet, which generates a menu of information and options based on the message's properties. The menu is divided into rows, with output icons on the left, explanatory paragraphs in the middle, and 1 or more action buttons on the right, as shown in Figure 3-2. The buttons execute code, and then switch to another applet depending on the action.

Preference actions switch to the preference page. The spam threshold preference is a slider that modifies the threshold variable. The menu page looks at this variable when deciding whether to add a flag icon indicating a message is spam.

Contact list actions switch to the contact list page. The contact list page has a toolbar at the top and the remaining area is divided into a contact list display table and a contact info/edit pane.

The contact list is displayed as a table whose table model is backed by a list of

contact objects. The contact objects store all their state in properties which can be accessed by string identifiers. Because all the property changes go through a single mutator method, it is an easy point for the contacts to trigger updates to anything that needs to change, such as the XML files in the current implementation. In the future, this could be replaced with a remote procedure call (RPC) protocol to update state on a remote server.

Incremental search is implemented by asking each contact whether it matches the search string every time the string changes. (If the previous search string was a subset of the new search string, then only the currently displayed list is searched. This allows the search to operate smoothly as the result list shrinks.) Each contact searches all of its properties to find if there is a match. The table updates to display the matching contacts. Clearing the search returns all the contacts to the view.

When a contact row is clicked, the side panel is linked to the contact data model. Edits to the panel's text fields update the contact data in real time. Thus, there is no need to click a save button before moving to the next contact or closing the window.

Clicking the delete button on a contact adds a "deleted" property to the contact object. This makes it display in a different color and enables the "restore" button which will unset the "deleted" property.

The advanced security options are hidden within a collapsible panel. This collapsible panel is a custom widget that resizes itself when its title bar is clicked. Its custom border offers a triangle pointing either up or down to indicate what will happen when it is clicked. The triangle highlights on mouse-over.

Chapter 6

User Interface Evaluations

The user interface went through an iterative design process. Three rounds of prototypes were tested on users, and the UI design was revised each time. This chapter details the results of these evaluations.

6.1 Briefing

Users testing Apuma prototypes were given the following briefing which tries to explain the purpose of the system without taking away the learning factor from the user's actual interaction with the system.

> This is a spam filter that adds on to your normal email client. It maintains a list of email senders you know and guarantees their email and only their email will be delivered to a protected inbox. It tries to maintain this list of senders as intelligently as it can without input from you, but if you are concerned that it will do the wrong thing, you will have an opportunity to control its behavior.

6.2 How do the users feel about the UI?

The underlying operational model of Apuma goes over the heads of most users. However, all the users were willing to use the system without understanding what was going on behind the scenes. Novice users are accustomed to using systems they don't fully comprehend.

Asking users to describe how they felt while interacting with the system revealed some important usability guidelines.

Users are most uncomfortable when they are presented with information they don't understand, and they feel that they are expected to understand it in order to proceed with their task. Users are comforted when confusing information is presented in such a way they they feel they can safely ignore it.

Hiding the Advanced Security Options behind a collapsing panel helps users avoid making uninformed decisions. When users are directly presented with an option, they feel they are expected to make a choice. Leaving things blank makes some users feel that they have failed to complete the task of configuration. Even if they have no idea what an option does, they will choose random settings. Putting a protective bubble around certain options gives users the feeling that is fine for them to leave those options alone, and in fact that it is a bad idea to mess with these advanced options unless they understand what will happen. Of course this only makes sense if the default value is truly a safe choice.

Having a configurable filter sorting incoming mail raises the question of whether changing a setting will cause previously sorted mail to move to a different folder. Some users said they expected it would, some expected it wouldn't, most did not know what to expect. This uncertainty is easily resolved with a little experimentation, although of course it should be made clear in the user documentation. There is a potential concern that users may think their settings changes are not working when they don't see any mail move, but all the test users who explored this seemed to presume correctly that the changes were simply not retroactive.

The style of contact list editing in which the editor state is kept in sync with the back-end is confusing to users who expect to have save and quit buttons, or ok and cancel. Users were initially worried about losing their changes. They did not seem bothered by the setup once they realized that all their edits were immediately saved. After closing and reopening a single time, this fact becomes obvious and the worry is negated.

When users deleted a contact and the entry turned to a disabled state rather then disappearing, the users expressed generally that it was not what they were expecting, but it made sense enough that they readily accepted it. In a future revision, I intend to try out a "trash-can" deletion model so that the deleted users are moved to a "deleted users" list where they will be out of the way.

A few users mentioned that they disagreed with the order of the columns in the contact list, but the preferred ordering was different between each user! Ideally, the columns should be draggable so the user can reorder them if they care to do so. This kind of variety in user opinions is a strong argument for total configurability whenever possible.

Chapter 7

Micro Payments and Micro Bonds

Designing a financial payment infrastructure is beyond the scope of this research. Apuma was designed so that any one, or many, payment infrastructures could be integrated. A system might not even need to use real money, as long as something of value was being risked and exchanged.

7.1 Requirements for a dedicated infrastructure

There are many ways a dedicated email payment or bond system could work. A protocol could be devised that would allow payment tokens to be issued by many banks, and cashed at other banks, similar to how paper checks function. This would allow competition and avoid centralized control, without fracturing the market and forcing users to choose between incompatible networks.

A payment should be able to be wrapped up in a text token that can be included in the email itself. Tokens should be unique so that a single payment can only be cashed once.

It is best if the payment token can be generated with no information about the recipient besides their email address. The address should be included so that intercepted emails cannot be harvested for money. The token should only be cashable by

the person whose address is in the token.

Encrypting the token with the recipient's key is an appealing idea, but since the payments are intended to ease first-time communication between previously unknown parties it would be a huge mistake to require the sender to acquire the recipient's key. Email addresses can easily be transferred verbally or on a handwritten note, while keys cannot. Until automated directory lookups can locate a key from within a mail client, it should be assumed that a sender may have the recipient's address but not their key.

This token could be a promise to pay, signed by the sender's key. The sender would not need to contact his bank to generate a payment, but the recipient's software should contact the bank to ensure available funds before considering the payment valid.

7.2 Building on existing infrastructure

Systems already exist to transfer small amounts of money from one account to another, the best known of which is PayPal, which allows single cent transactions. By putting a few dollars into a PayPal account, a sender can make many small payments without paying credit card transaction fees each time.

One possible method to tie these payments into email would be to generate a hash of the message, and include it in the PayPal payment comment field. The recipient could verify the match between the email and their PayPal transaction history. This could all be handled automatically, since PayPal provides an API for programmers to interface directly to their network. The payments could be considered conditional, because the recipient can easily send back the money if the message was desired.

7.3 Security concerns and recommendations

7.3.1 Risk of financial loss to the user

Using micro-payments in email raises the same security concerns as any other e-commerce. A trojan program infecting an unsuspecting user's computer could key capture their password and then send out spam with stolen micro-payments. Since it is already known that spam is being sent from compromised machines, this scenario is likely to occur if payments become required for mail.

The best solution I have thought of is to limit the rate of outgoing payments for a particular account. There is no general reason a person would need to send more than a small handful of payment-backed messages in a day. If a specific reason were to occur, the sender could call their bank to authorize a temporary limit increase. This is different from the limitations imposed by a system like HashCash, where the user physically cannot exceed the maximum sending rate. HashCash would require a legitimate bulk sender to invest in expensive computer hardware; the system I am proposing could easily accommodate senders who desire accounts without limits, if they are responsible for securing their systems against trojans.

It is also very important that transactions be recorded in a way that the user can easily audit their expenditures. In case of fraud, the payments should be reversible.

7.3.2 Risk of exploits that allow spam

Any email payment system should ensure that a legitimately generated payment cannot be hijacked and used to send spam.

An unsigned message could be modified, and its content replaced with spam. Signing messages prevents them from being modified in transit. The payment token could also include a hash of the message, so that it could not be applied to a different message.

The protocol should assume that the email travels over an open channel, and that

an attacker could see the message and make use of a stolen payment before the real message arrives.

7.4 Choosing threshold values

The user should choose a threshold value that is high enough that they feel compensated for the annoyance of ads, but low enough that it would be no concern for a real person to risk that sum to contact the user.

Chapter 8

Conclusion

Maintaining the open nature of email requires putting control into the hands of the recipient. Giving them control without overwhelming them with micro-managerial chores requires an intelligent self-maintaining system. The Apuma system described in this thesis was designed and implemented to be such a system.

The desire to filter 100% of spam, and the need to avoid the loss of legitimate mail creates a need to identify emails that should not be subjected to filtering. Apuma attempts to solve this problem using intelligent rules to maintain an accept-list.

Messages from known senders can be exempted from filtering, eliminating the vast majority of potential false-positives. Remaining mail can thus be subjected to much more rigorous screening.

Special cases such as first time contact can be handled with micro-payments or micro-bonds.

Apuma includes a plugin interface that allows any financial, proof-of-work, or other desired protocol to be integrated into the Apuma filtering framework.

8.1 Contributions

The primary contribution of this thesis work is the implementation of a plugin-based spam filtering framework. The implementation of the Apuma system addresses questions of how to store and edit contact data in ways that are clear to the user, how to incorporate any plugins into the processing framework, and how to keep an accept-list updated by monitoring incoming and outgoing mail.

This thesis also presents and argues several ideas about how email should function: only the recipient has the right to discard a message; email should be signed to prevent impersonation; sending unwanted email to a stranger should cost money, and furthermore the money should go to the recipient; and decentralized, non-proprietary infrastructure is more trustworthy and ultimately more useful than centralized infrastructure with proprietary protocols.

Chapter 4 presents a carefully designed ruleset, which attempts to maximize the ways a legitimate message can get to the recipient, without opening any holes for spam to slip through. This chapter also presents auto-managed accept-lists as a powerful tool to protect children's inboxes without impinging on their freedom.

Chapter 7 proposes several ideas for micro-bond / micro-payment systems that would integrate either existing financial infrastructure or new open-standard protocols into Apuma's framework.

8.2 Future work

The problem of spam management, and intelligent email handling in general, is still very much unsolved. The problem of keeping spam off mailing lists is very complicated, and cannot be totally handled by the client. Lists must have carefully thought out policies controlling who can and cannot post, and what kinds of posts are allowed. A non-moderated list, fully open to the public, is rarely, if ever, necessary. Most lists could easily restrict posting access to a set of domains, or to the list membership

itself. I further suggest that mailing list software should maintain its own signing key, and sign messages that it has approved. Then clients could trust this key, and would not have to worry about managing an accept-list of individual senders for list mail.

Spam, or unwanted advertising in general, is a problem that extends way beyond email. As open communication and user-generated content become more pervasive, through the use of blogs, guest-books, forums, instant messaging, and internet cell-phones, we must find ways to pull the relevant content out of the noise. In doing so, we run the risk of creating censorship, or silencing the individual voice. We should always be on the lookout for any idea that can place control in the hands of the users, while taking away the burden of micro-management.

If proof-of-work tokens were of some use to the recipient, or contributed to something the recipient cared about, they could qualify as a sort of micro-payment. There are some distributed computing projects running today. This gap can be bridged if a way can be found to show proof of a contribution to such a project, and use it as a proof-of-work token.

Apuma will benefit from plugins for any protocol that should affect whether a message gets delivered. Systems like IronPort, HashCash, and Habeus would be good candidates for Apuma plugins.

Apuma processing trees must currently be written by hand. It would be very helpful to have a drag and drop flowchart editing program that could output Apuma configuration files.

Apuma was intentionally designed to be mail-client independent, but better usability results could be achieved by integrating the message annotation directly into a mail client. The ideas of Apuma could also be integrated very tightly into webmail systems.

Bibliography

[1] AOL Whitelist Guidelines. http://postmaster.aol.com/guidelines/sender.html.

[2] Barracuda Networks. http://www.barracudanetworks.com/.

[3] Habeas Sender Warranted Email. http://www.habeas.com/.

[4] IronPort Systems. http://www.ironport.com/.

[5] Mailblocks. http://about.mailblocks.com/.

[6] Personal communication, Northland Cable Tv mail administrator. http://www.northlandcabletv.com/.

[7] Sender Policy Framework. http://spf.pobox.com/.

[8] Spammotel. "http://www.spammotel.com/".

[9] The Apache SpamAssassin Project. http://spamassassin.apache.org/.

[10] The International PGP Home Page. http://www.pgpi.org/.

[11] The Spamhaus Block List. http://www.spamhaus.org/sbl.

[12] Thunderbird, reclaim your inbox. mozilla.org. http://www.mozilla.org/products/thunderbird/.

[13] Vipul's Razor. http://razor.sourceforge.net/.

[14] Adam Back. Hashcash - Amortizable Publicly Auditable Cost-Functions. Technical report, August 2002. http://www.hashcash.org/.

[15] Jo Best. Microsoft bankrupts the Spam King. ZDNet, March 2005. http://news.zdnet.co.uk/internet/0,39020369,39193140,00.htm.

[16] Liane Cassavoy. Mailblocks Finds Its Messages Blocked. PC World, November 2004. http://www.pcworld.com/news/article/0,aid,118446,00.asp.

[17] Michelle Delio. Not All Asian E-Mail Is Spam. Wired, Feb 2002. http://www.wired.com/news/politics/0,1283,50455,00.html.

[18] Technology Directorate For Science, Computer Industry Committee For Information, and Communications Policy. Background Paper For The OECD Workshop On Spam. Technical report, Organisation for Economic Co-operation and Development, January 2004. http://www.olis.oecd.org/olis/2003doc.nsf/LinkTo/dsti-iccp(2003)10-final.

[19] Federal Trade Commision. You've got spam: How to "can" unwanted email. http://www.ftc.gov/bcp/conline/pubs/online/inbox.htm.

[20] FTC. Controlling the Assault of Non-Solicited Pornography and Marketing Act of 2003 (CAN-SPAM Act), 2003. http://www.ftc.gov/bcp/conline/edcams/spam/rules.htm.

[21] Yu han Chang. Email Filtering: Machine Learning Techniques and an Implementation for the UNIX Pine Mail System, May 21 2000. http://citeseer.ist.psu.edu/454438.html; http://www.ai.mit.edu/~ychang/learningmail.pdf.

[22] David Heckerman, Eric Horvitz, Mehran Sahami, and Susan Dumais. A Bayesian Approach to Filtering Junk E-

Mail, May 06 1998. http://citeseer.ist.psu.edu/126936.html; http://research.microsoft.com/~sdumais/spam98.ps.

[23] IETF. S/MIME Mail Security. http://www.ietf.org/html.charters/smime-charter.html.

[24] Philip Jacob. The Spam Problem: Moving Beyond RBLs, 2002. http://theory.whirlycott.com/~phil/antispam/rbl-bad/rbl-bad.html#collateral.

[25] David Heckerman Joshua Goodman and Robert Rounthwaite. Stopping Spam. Scientific American, April 2005. http://www.sciam.com/article.cfm?chanID=sa006&colID=1&articleID=000F3A4B-BF70-1238-BF7083414B7FFE9F.

[26] Ben Laurie and Richard Clayton. "Proof-of-Work" Proves Not to Work. http://www.apache-ssl.org/proofwork.pdf.

[27] Brian McWilliams. Spam, the Nazi hunter and Citizen Joe. Salon, September 2004. http://www.salon.com/tech/feature/2004/09/07/spamfight/.

[28] MessageLabs. Email threats overview. http://www.messagelabs.com/emailthreats/.

[29] Microsoft. Sender ID Framework. http://www.microsoft.com/senderid.

[30] Moni Naor. Verification of a human in the loop, January 20 2002.

[31] Sarah Nathan. AG Reilly Sues Deceptive Spammer for Violating Massachusetts Law, Federal CAN SPAM Act. The Office of Massachusetts Attorney General Tom Reilly, July 2004. http://www.ago.state.ma.us/sp.cfm?pageid=986&id=1257.

[32] Steve Outing. I'm Sick and Tired Of Spam (Filters), June 2002. http://www.editorandpublisher.com/eandp/news/articledisplay/article_display,jsp?vnu_content_id=1570036.

[33] Philip R Raymond. System and method for discouraging communications considered undesirable by recipients. US Patent 6,697,462, April 2002.

[34] Ron Rivest. Peppercoin micropayments. In *FC: International Conference on Financial Cryptography*. LNCS, Springer-Verlag, 2004.

[35] Chilin Shih, Daniel Lopresti, and Greg Kochanski. A Reverse Turing Test Using Speech, January 27 2003.

[36] Luis von Ahn, Manuel Blum, and John Langford. Telling Humans and Computers Apart Automatically. *Communications of the ACM*, February 2004.

[37] Alma Whitten and J. D. Tygar. Why Johnny can't encrypt: A usability evaluation of PGP 5.0. In *8th USENIX Security Symposium*, 1999. citeseer.ist.psu.edu/whitten99why.html;citeseer.nj.nec.com/whitten99why.html.

[38] Yahoo. DomainKeys: Proving and Protecting Email Sender Identity. http://antispam.yahoo.com/domainkeys.