Fighting Spam by Encapsulating Policy in Email Addresses

John Ioannidis
ji@research.att.com
AT&T Labs – Research

Abstract

Everyday network interactions require users to give out their email address, yet no guarantees can be made about how this address will be used. Sometimes the address is given to a human (e.g., on a business card), but many times it is entered in a web form as part of a web transaction. Unfortunately, that email address frequently finds itself in spammers' lists, never to be removed again. We propose the concept of the Single-Purpose Address (SPA); SPAs are generated by a program, and then cut-and-pasted into the web page (or other application) requesting an email address. In its more general form an SPA encodes a security policy that describes the acceptable use of the address; since senders cannot be trusted to abide by it, this policy is enforced by the receiver, who generated the policy in the first place. SPAs are cryptographically protected to shield them from tampering. Since SPAs are not meant to be typed by humans, but rather processed by computer only, they do not have to be short, memorable, or even pronounceable. We present ways to construct such addresses, and we show the implementation of an optimized case.

1. Introduction

Unsolicited commercial email ("spam") is everywhere. Much effort is being spent to curtail it, and no arguments need to be offered here as to why spam is undesirable. While there is no perfect solution in sight, decreasing exposure and increasing the trouble that a spammer has to go through to send unsolicited email seems to be an acceptable approach. If only a small number of users employ these methods, it will not be worth it for the spammers to try to work around them; if many people start using them, the cost to the spammers of working around so many protections may make it financially unrewarding to engage in their practices.

The life-cycle of spam starts with our making our email address known by giving it out to individuals, subscribing to mailing lists, posting on Usenet or other on-line discussion media, entering it on web forms when conducting web-based transactions, and so on. Sometimes the email address we give out needs to be parsable by humans; that is the case with email addresses we give to friends, colleagues, business associates, and other people we interact with personally. In many more cases, it is irrelevant if an address is simple and readable (*e.g., ji@att.com*) or completely obscure (*e.g., VP72W24KM7IH7FT4O@fufutos.gr*). These are the cases when the address is not processed by a real person (that is, read and then typed in in the course of everyday transactions), but instead is passed between databases. It is precisely then where it is important to be able to limit the use of an address we gave out to just the purposes for which it was given out. It is generally believed that most spammers collect new email addresses by scanning online postings, and by buying or otherwise acquiring lists of customer or personnel addresses. In all of these cases, the readability of the address matters little, since no human typist is involved. It can even be argued that, with modern mail user agents (MUAs) offering sophisticated GUIs allowing users to select addresses based on attached real names rather than actual email addresses, and also people exchanging virtual business cards by pointing their PDAs at each other, even humans may not need to be able to read and remember an email address.

We can see the following broad (and usually overlapping) categories of email address exposure:

1. Person-to-person, where communication is explicitly invited.
2. Online presence, where communication by readers of the online material is welcome, at least for a while.
3. Electronic transactions, where communication is only desired when initiated by the other parties involved in the transaction.

In the first case, where we give our address to a human, we want that address to work for a long time, perhaps forever, but we would ideally want only that person to be able to send mail to it. This may be feasible if we employ some electronic means of giving out the address (more email, a

PDA), but is probably impossible in the general case for email addresses printed on business cards, for example.

The address-on-the-card is just a special case of the second type of address exposure; it is similar in nature to putting in on a research paper such as this one, posting on Usenet, and so on. For example, if I had given my email address on the title page as *jiREMOVE@THE.research.CAPS.att.com*, any human reading the paper would immediately figure out how to send me email, but a web crawler getting the electronic version might have a harder time. Many such address obscuring techniques are employed; we list some of them in Section 5. The problem here is that, in some venues, we may want the address to work indefinitely, whereas in other cases we want it to have a limited lifetime; either way, we cannot restrict who the sender will be, except that we hope that, by obscuring it, they will be a human.

Let us now examine the third case. In many situations of conducting business over the web, we only care that the corresponding party be able to contact us for a limited amount of time; moreover, we want *only* the organization we are giving our address to to be able to contact us, and not anyone they pass it on to. This is both to prevent said party from sending advertising material in the future (which most online vendors do, despite assurances to the contrary), and to prevent abuse of the supplied address by third parties that, with or without the coöperation of the merchant, acquire our email address. Thus, the two main features required of email addresses in these situations are that they stop working after a certain date, and also that they only work for specific senders.

An easy approach would be to create throw-away email aliases, give those out, and not check mail sent there after a while to the ones we consider to be single-use or limited-time-use only. It is easy to see that such an approach has huge logistical overheads. Also, most Internet users do not run their own mail servers, and creating aliases at a moment's notice is not an easy option for them; the *username+extension* convention[1] can be used as a form of alias in this case. After an alias has reached the end of its desired lifetime, a filter rule in a feature such as *procmail* can be used to reject the alias from then on. Evidently, *procmail* can also be used to restrict who the allowed sender for that particular alias is. An unfortunate problem with this approach is that many web servers do not accept email addresses containing a + sign. Also, since the part before the + sign is still presumably a valid address, the next generation of spamming software would simply remove whatever followed the + and send to the base account.

Many other ad hoc solutions can be proposed, and all of them have the problem that the size of the alias list and the size of filtering rules grow without bound, since entries can never be removed. What we propose instead is to *encode* rules as part of the email address, and do it in such a way that the potential senders cannot alter these rules without at the same time invalidating the alias. These rules are applied when email to the address is received. This way the user does not have to store any per-address rules locally or keep track of multiple email addresses. In Section 2 we describe the general way to accomplish this goal; in Section 3 we present the implementation of a common case; we discuss further aspects of the system in Section 4. We conclude with related and future work.

2. SPA Architecture

Figure 1 shows the players in the SPA system. *Users* create *SPA*s which they give to *correspondents*. When correspondents send email back to users, either the user's MTA (Mail Transport Agent) and/or the user's MUA (Mail User Agent) check the email against the policy rules encoded in the address, and deliver it, bounce it, or discard it accordingly. This, of course, is a very simplified view of the mail delivery process; it does not take into account the possibility that a remote mail access protocol such as POP3[13] or IMAP4[8] is used. We shall discuss these cases in Section 4.

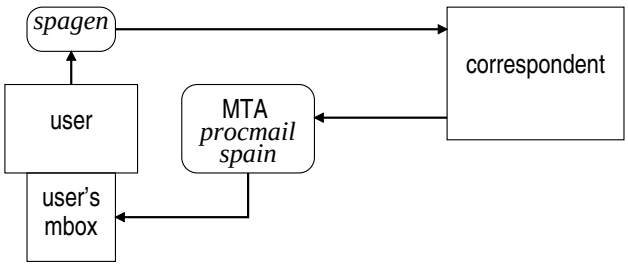

Figure 1. **Give address get mail.**

The core idea is that whenever an email address is handed out, rules about its use are encoded in the address itself. The correspondent cannot, of course, be trusted to obey these rules; the rules are enforced by the MTA and/or the MUA when mail is sent to that address. The correspondent may also attempt to tamper with the structure of the SPA in an effort to cause a message to be delivered when it should not be; thus, SPAs should contain information that allows their integrity to be verified. Furthermore, there is no reason to divulge any information about a user's policy to the correspondent; the address must therefore be encrypted under a key known only to the agents who generate the SPA (the user) and the agents

[1] In most email systems, sending mail to *user+extension@domain* delivers the mail to a "mailbox" named *extension* for *user@domain*. The mailbox can be a separate mail file, or can simply be a marking of the mail message indicating that it is for that particular mailbox.

who must process the message when it is received (the MTA and/or the MUA).

The requirements for the system we are building are:

- Any user should be able to set it up with little or no explicit support from the mail administrators.
- Users should not have to get their own domain names, or manage their own mail servers.
- The addresses should look like ordinary (if unpronounceable) email addresses to correspondents.

2.1. Structure of Single-Purpose Addresses

The SPA consists of two parts: an indication of the addressee, and an appropriately-encoded description of the policy that will be applied when the message is received. The addressee can simply be identified by his username, with the policy part given as the extension as in the *user+extension* convention. Since presumably the "naked" (with no extension) main address of the user would still be valid, it is recommended that users who want to use SPAs get a second address (or an alias), and set up their systems so that mail to the naked second address is rejected. We shall discuss this more in Section 3.2.

The policy is a set of rules that encode the conditions under which the sender's email is allowed to be delivered to the user or not, and what corresponding actions should be taken. This definition of policy is much like in one in trust management systems[7, 11]. A policy could be as simple as "*this email address expires on March 20th, 2003"*, or as complicated as "*accept this mail between January 30th, 2003 and March 20th, 2003, and only if the user is sending it from some machine in cs.miskatonic.edu; if accepted, forward the mail to seldon@trantor.gov*." What is expressible is limited by the richness of the policy definition language, and how compactly it can be encoded. The SMTP specification[14, 10] does specify a 64-character size for usernames, but modern mailers seem to support much longer addresses. No one is expected to type in such addresses; remember that SPAs are meant to be generated by a program and cut-and-pasted into web forms or other such places.

Evidently, email headers are not authenticated in the normal course of events, hence no guarantees can be made about their validity. It is not the objective of our system to supplant secure email; rather, it is one more tool in the spam-fighter's toolbox. The one part in the SPA that can be trusted is the expiration date, and that is because it is checked against the receiver's notion of time, who decided on the expiration date in the first place. This also avoids the need for synchronized clocks between sender and receiver. The SPA has to be protected against tampering by the correspondent, and even against examination, so that headers cannot automatically be faked to match the policy in the address. Once the set of policy rules has been assembled, it is MAC-ed and encrypted under (symmetric) keys known only to the user creating the SPA (and subsequently receiving mail sent to the SPA, of course). The details of cipher selection are beyond the scope of this paper, but we do justify our implementation selections in Section 3.1. Care must be taken so that individual bit changing or splicing attacks do invalidate the MAC.

The output of the encryption is a string of random-looking bits, and as such it is not suitable for use as an email address. It must therefore be encoded using a set of characters that are legal for email addresses[14, 9]. After the policy part has thus been obtain, it is appended to the user part; the result can now be cut-and-pasted or otherwise used as an email address.

2.2. Processing of Received SPAs

When email directed to an SPA is received, the SPA must first be decoded. The username part is stripped, and the remainder is decoded and decrypted. Next, the MAC is checked; if it passes, the policy rules are evaluated within the context of the user and of the received message (*i.e.,*, if the rules refer to the user's environment such as the date, or parts of the message such as the sender address, the program that checks the rules has to take that into account). The rules could indicate a simple accept/reject choice, or may have more complicated results such as a new address to forward the mail to, extra headers to insert, and so on. Typically, all these operations will be done by a program invoked by *procmail* or other mail delivery processing applications.

An impediment to the proper operation of SPAs is that, in many legitimate cases and in most spam cases, the address listed in the *To:* header of the mail message is not the same as address of the recipient, which is given in the *RCPT* command (the so-called "envelope recipient address") during the SMTP dialog. In fact, in many cases the envelope recipient address does not appear anywhere in the headers, although some mailers, if properly configured, will put it in one of the *Received:* header lines. There is only one solution to that: the MTA where the SPA is delivered must be configured to add a header with the envelope recipient address, which, of course, is the SPA. There are issues when multiple recipients are given with *RCPT* commands; we discuss how to solve them in a particular MTA (Postfix) in Section3.2.

3. Implementation

We implemented SPAs under Unix, with Postfix[15] as the Mail Transport Agent. There are two phases; creating the SPA, and then processing email sent there. Unlike the

previous section where mostly talked about the structure, in this section we shall discuss the receiving considerations much more than the creation of SPAs.

3.1. SPA Creation

As a first usable system we decided to limit the range of policies expressible in the address for the sake of brevity and usability of addresses. There is a lot of broken and misconfigured mail software and we would like, if possible, not to cause it to reject our addresses. Given that we want these addresses to be used for web-based transactions, and that some web servers impose artificial limits on the size of email addresses, we decided to put the least amount of data, encoded in a compact way, that fulfilled the design criteria of our system. To wit, these baseline SPAs have only an expiration date and a substring of the domain name that is allowed to send email to that address.

A fundamental design issue was the length of the SPA. 128 bits are a convenient size; they are one block of a modern symmetric cipher such as AES. We shall call this construct the SPAB (SPA Block). SPABs are encrypted under a symmetric key, and encoded using the character set of valid email addresses: letters, numbers, and a few special symbols. The encrypted and encoded SPA block is called the SPABEE. Base-64 encoding would be very convenient, resulting in a 22-character-long SPABEE. While there is no reason to assume that automatic software would change the case of an SPA string, email addresses are case-insensitive, so we decided not to take any risks and just do base-32 encoding, using the numbers 0–9 and the letters A–V. On receipt, we ignore the case of letters. This results in a 26-character-long SPABEE. A quick experiment of various well-known web sites showed that they had no problem with 40- or 50-character-long addresses.

Time resolution of one day was deemed adequate; allocating 14 bits for the expiration date, encoded in days since January 1st, 2000 would allow for expiration dates into September 2044, by which time it is hoped that we will have eradicated the spam problem. The special value 0 means "no expiration date". Some bits in the 128-bit string should be used to check for integrity, and the rest of the space should encode the domain name allowed to send to the SPA. Two bits encode how many trailing domain name components to strip from the incoming address before comparing it to the address in the SPA. The underlying assumption is that, in many occasions, the significant part of a domain name (as far as spam is concerned) does not include the trailing gTLD or ccTLD, and some times not even the second-level domain. We want a mechanism for allowing "foo.com", "foo.co.uk", and "foo.ac.jp" to all be accepted if we suspect that they are the same firm. The first two bits in the SPAB encode this: "00" means that nothing is stripped, "01" and "10" means that one or two trailing domain components are stripped, and "11" means that upon receipt we should try matching against zero, one or two components stripped. The domain name itself can also be encoded in 6-bit characters, since domain names are not case-sensitive, and many characters are not valid domain name characters. If the domain name is shorter than the space available, it is padded with zeros. Of course, an all-zeroes domain part means "accept all domain names". Given all these considerations, we come up with the following encoding:

2	bits encode TLD stripping.
14	bits encode expiration date (to September 2044).
96	bits encode domain name (16 characters).
16	bits encode ICV.

Since the domain name can now encode 16 characters, we can even have it encode a full email address (the fact indicated by the inclusion of an at-sign in the allowed domain name). Many people's email addresses are longer than 16 characters; again, we use properties of the application domain to circumvent this problem; if an @ sign appears in the domain name, we make a substring comparison anchored at the @ sign. For example, *"i@research.att.c"* would match *"ji@research.att.com"* (as well as *"iii@research.att.co.uk"*), but would not match *"abc@research.att.com"* or *"ji@yahoo.com"*.

The ICV is derived by computing the MD5 checksum of the first 112 bits, and using the first (high-order) 16 bits of that checksum. This is not really a MAC, but it is adequate. It may seem rather short, however, in that it would allow one tampered message in 2^{16} to go through undetected. This is not deemed to be a serious problem; for most people, it would be less than one bad message per month, and even then, the chances that the domain name and the expiration date would have decrypted to something valid are virtually nil.

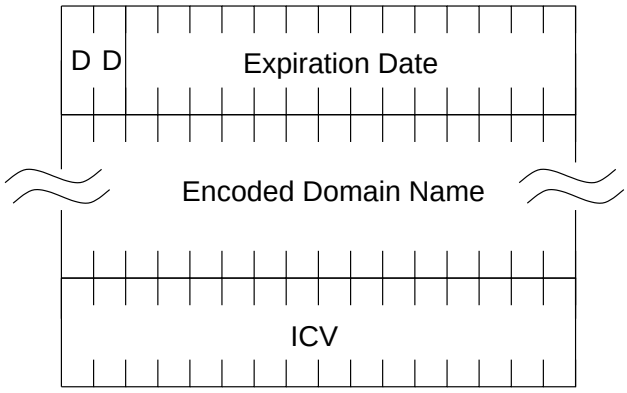

Figure 2. **SPA Block.**

The format of the SPA Block is shown in Figure 2.

To generate it, we wrote a short script, *spagen*, which takes as arguments an expiration date and/or a domain name (with or without an @ sign, as described above), computes the ICV and populates the 128-bit SPAB. The resulting block is then encrypted with AES-256-CBC (AES with a 256-bit key in CBC mode), and printed out base-32 encoded. The contents of the environment variable SPAPFX are prepended:

```
$ echo $SPAPFX
ji+QQ
$ spagen -e +4 -1 -d apskatecomputers
ji+QQIV89O7OD2GDASIK2M1928IN6N8
```

This string can now be cut-and-pasted into whatever application requires an SPA. The `-1` option indicates that one level of domain names has been omitted; the generated SPA will therefore cause any mail originating at *cheapskatecomputers.com*, as well as *realcheapskatecomputers.org*, to be accepted for the following four days.

Once the SPAB, the policy part of the SPA, has been computed, and encrypted and encoded into the SPABEE, the SPA can be formed in several different ways, depending on how the user's receiving email software operates. The easiest case is when we are only worried about mail sent directly to us; then we can form the SPA using the mailbox+extension convention: the username, followed by +, followed by the SPABEE. Since SPABEEs are 26 characters long, and it is unlikely that other ordinary uses of the mailbox extension will be that long, a *procmail* rule that selects 26-character extensions can be used to divert email sent to SPAs. Alternatively, some user-selected prefix to the SPABEE can be used as a disambiguator. In the previous example, the SPABEE can be prefixed with the string "QQ"; then a *.procmailrc* rule like the one in Figure 3 be used. All incoming mail to *ji* with an extension starting with *QQ* will be piped to a program called *spain* ("SPA in") for processing.

```
:0
^TOji+QQ.*@att.com
| $BINDIR/spain
```

Figure 3. **.procmailrc rule to catch SPAs**

3.2. Processing incoming SPA mail

The simplest case for processing SPA mail is when the "+" convention is used. The user's *.procmailrc* contains a rule like the one in Figure 3. The mail is piped through the receiving program (*spain*), which verifies the validity of the SPA. If the SPA is invalid, the mail is simply discarded. Otherwise, the program inserts an *X-SPA-From:* header line into the incoming mail and pipes it back through procmail for reprocessing. Any existing *X-SPA-From:* lines are deleted. To prevent an adversary from inserting random *X-SPA-From:* lines in their mail in an attempt to bypass SPA processing, *spain* replaces the `To:` header line with *username*+ a fixed 128-bit string (base-32 encoded). Mail sent to this extension is known to have been sent to a valid SPA, and can take the *X-SPA-From:* header into account. This process is adequately secure; the security of the system is predicated upon the filesystem security of the user anyway (anyone who can break into the user's machine and/or mail repository can simply deposit spam there). Since this secret delivery address is never exposed, and it only appears in the user's configuration files, there is no additional security exposure. An adversary has as much of a chance of guessing that address as of guessing the encryption key, which is negligible. Obviously, mail re-injected this way will not match the rule looking for SPAs. Rules that do match it can then also look at the *X-SPA-From:* headers, along with other headers, to split the mail into different mailboxes. Figure 4 illustrates this process.

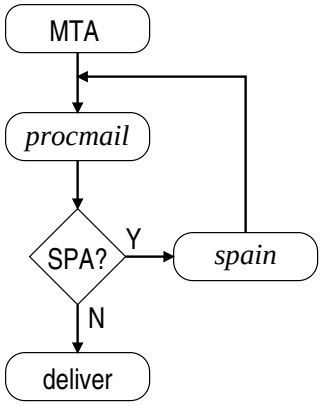

Figure 4. **Incoming SPA mail.**

A number of complications arise in practice, all of which need some (usually just one-time) support from the mail administrator. The first, and perhaps most significant, is that many web form software does not accept a + sign as part of the email address; whether that is by design or by ignorance is immaterial. Although MTAs can be configured to use a different extension separator character, that may not be desired because it affects all users. A better solution is to use a separator string, such as *QQ*, that is not likely to appear as part of an email address. For example, if the mail server for *fufutos.gr* uses the *postfix*[15] MTA, the following rule in */etc/postfix/virtual_pcre* accomplishes this.[2]

[2]For this to work, `pcre:/etc/postfix/virtual_pcre` must be added to the `virtual_maps` line in */etc/postfix/main.cf*.

```
/^(\S+)QQ(\S+)@fufutos\.gr$/
   ${1}+${2}@fufutos.gr
```

Then, any mail sent to, *e.g.*, *fooQQbar@fufutos.gr* will be delivered to *foo+bar@fufutos.gr*.

This still leaves open the possibility that someone perusing a list of spam addresses will notice the pattern and decide to simply put *foo@fufutos.gr* in the spam list. To solve this problem, each user can be given an alias for use only with SPAs. Mail sent to the naked alias (without an extension) is obviously spam, and can safely be rejected.

The other important operation that the SMTP daemon has to perform is to save the envelope recipent address (the address given in the SMTP *RCPT To:* command); the reasons for this were explained in Section 2.2. It turns out that this is harder than it looks, because it has to happen before any aliasing or address rewriting happens in the MTA. The easiest way of accomplishing this in Postfix is by defining a "content filter" for *smtpd*, the program that accepts incoming SMTP connections. If so configured, *smtpd* runs the mail it gets through a filter, giving the filter access to the envelope sender (*MAIL From:*) and receiver (*RCPT To:*) addresses. Thus the filter can insert a header line such as

`X-SPA-To:fooQQIV8...928IN6N8@fufutos.gr` (where some characters have been omitted for formatting reasons) to the headers. Because spammers can always insert their own *X-SPA-To:* headers, the filter scans for them and removes them before inserting its own.

After these substitutions, the MTA delivers the mail to the final user. The rule in Figure 3 is duplicated, except that the header matched is not the *To:* header but rather the *X-SPA-To:* header. Otherwise, the processing that invokes *spain*, checks the SPA, and resends the mail to the user with the 128-bit secret extension as described in the beginning of this section remains the same.

4. Discussion

There are several operational issues associated with the use of SPAs. The first concern is what happens when a spammer acquires an SPA, but also learns who the allowed sender is. Since email headers are not authenticated at any point, the spammer can simply forge the *From:* headers and let their spam through. Similarly, unrestricted SPAs given to friends may also be compromised, for example by being given by a naïve relative to an electronic greeting card web site. We thus need a way to revoke SPAs without actually having to keep track of all the ones we have given out. The easiest way to accomplish this is to use periodically change the encryption key, while keeping some small number of old ones around. When mail comes in that does not decrypt properly with the current key (the ICV in the SPAB does not compute correctly), we can try decrypting with older keys; if one of them works, depending on the policy in the SPA we may simply consider it expired, or send a challenge to the sender to verify that they are still legitimate. The challenge message should not create any additional state; an easy way of accomplishing this is to generate an SPA with a one-day lifetime asking the recipient to reply to that address. Many variations on this theme are possible, and experience with a deployed system will show what is most appropriate.

The 128-bit SPA described in Section 3 was done as the absolute minimum system that would have the shortest possible SPABEE size. In fact, since we are only checking for equality, the string itself need not be in the SPAB, just an indication of how many characters before and after the @ sign to compare, along with a truncated MAC of the resulting substring. This would allow very long email addresses to be efficiently encoded, but would not reduce the minimum size of the SPABEE, which is dictated by the block size of the cipher. If we are willing to allow SPAs longer than 128 bits, we can encode longer acceptable email addresses, perhaps even use real regular expressions instead of substring matching. We can also add a forwarding rule: an email address to forward the resulting email to, in order to make *procmail*'s job easier. We can also increase the granularity of the expiration date to hours or minutes, but since email operates on human scales, the benefits derived from that would be almost nonexistent. There are some more obvious policies that we could encode with a single bit, such as requiring encrypted/authenticated mail to the SPA. In the limit, full-fledged policies can be expressed in some policy definition language such as the one described in [6].

Finally, when systems such as POP3[13] or IMAP4[8] are used, the user can have the option of letting all mail, including spam, accumulate on the server, and then use *spain*-like processing as each piece of mail is retrieved. Naturally, usually only the headers will need to be retrieved before a decision is made to discard the mail as spam. If the machine where the POP3/IMAP4 server runs can be trusted with the user's SPA key, then all this processing can happen on the server side, eliminating the need to keep spam in the user's mailbox.

The basic premise of Single-Purpose Addresses is that they do not rely on the coöperation of the correspondent to enforce policy. In trying to extend the system, we should keep that basic premise in mind. There are many ways to fight spam; SPAs add one more powerful tool to our armory.

5. Related Work

There is a tremendous amount of effort being exerted to fight spam. Like any social disease, people have tried to counter spam with social pressure, technology, and legis-

lation. The social pressure aspect is the weakest of the approaches, and only works to some extent when there is the possibility of financial or societal repercussions. Some legal framework exists that prohibits unsolicited email. [12] is a not-very-recent survey, and legislation by region can be found in [1]. Litigation creates problems of its own, and it is usually easier to just tolerate spam as part of the cost of doing business on the Internet.

The main focus of anti-spam combat has been through technical means. Again, much like disease, there are ways of reducing exposure in the first place, combating the way it spreads, and identifying and removing it where possible. Reducing exposure is hard, and different techniques need to be employed when giving addresses to individuals, posting in online fora, and when conducting transactions over the web. These categories are not even well-defined; a user entering a friend's email address in an electronic greeting card web site thinks that he is being cute, but he is are actually exposing his friend's unrestricted email address to a spam harvester. Moreover, in personal email it is hard to use one-time addresses; social norms still dictate the use of short, pronounceable email addresses.

The system most like SPA is the Tagged Message Delivery Agent (TMDA)[2]. TMDA also has the concept of using the email address to create single-purpose addresses. Aside from formatting and implementation details, the main architectural difference is that policy is not explicitly described in the email address, but rather (except for the case of the expiration date) the address is used to look up the policy in local tables. This means that for each special address created, state must be kept by the user so that it can be processed in the future, causing such state tables to grow without bound when addresses without expiration dates are used. TMDA also includes features found in challenge-response systems. These systems, (*e.g.*, [3]) try to ensure "return routability" of email addresses, potentially including a short puzzle that a human could answer easily but a machine could not. These can mitigate the problem of spam to harvested addresses, but it does mean that there can be cases where an automatic email should have gone through but it does not. SPAs address this problem.

Much spam software is sloppy about following SMTP to the letter. Other software uses "open relays" (MTAs (mis)configured to accept mail not for their own domains), as stepping stones to obscure the origin of spam. Several services provide real-time data on whether a host is an open relay, and modern MTA software includes extensive support for querying these services and rejecting email passing through an open relay. Similarly MTAs can be configured to reject mail with badly-formed headers; unfortunately, a sizeable fraction of worldwide mail servers are misconfigured, and the risk of losing important email is high.

Various ad hoc solutions also work, such as seeding spammers' lists with email addresses created just for that purpose, and comparing mail received on those addresses with mail received on real addresses. Spammers have been getting more insidious, though, and they generate slight variations of the message to each address they send to.

Finally, there is a large amount of research literature on identifying spam through textual analysis of both the headers and the body of the message. Spamassassin[4], a message-text classification program, does a stellar job, as do many other systems employing AI techniques; their operation is beyond the scope of this paper. Even forms of micropayments have been proposed, either in the form of actual transfer of payment, or the burning of CPU cycles as an indication that some expense has been incurred[5]; causing spammers to spend their own resources in a manner proportional to the amount of spam they send may make them rethink their business model.

6. Summary and Future Work

There several ways in which to expand the SPA idea. A policy description language that is more expressive than simple regular-expression matching, and that can refer to other headers and perhaps even content of the message may be a useful tool to develop. Another useful extension would be to enable a mail server along the delivery path, rather than the end user, to apply the policy in the SPA. A fairly obvious way of accomplishing this is to encrypt the policy under the server's public key while signing it with the user's key. This of course opens up the potential to launch a CPU denial-of-service attack on such servers, hence some details have to be worked out first. Finally, a method for delegating the right to send email may be of some limited use: a subscriber to a mailing list may wish to let any member of the mailing list send him mail without first contacting him. If the SPA encodes that the mailing list administrator has the right to delegate to others the ability to send mail to the user, and the sender of such email includes a delegation certificate along with her email, that email will be let through. Many policy management systems, such as [6], have a natural way of specifying delegation. Such delegation certificates can be transmitted in additional mail headers.

In conclusion, we have described *Single-Purpose Addresses*, a system that encodes policy in email addresses for the chief purpose of filtering spam intelligently. The policy specified may be as simple as an expiration date and/or the email address of the allowed sender, or as complicated as a full-fledged program. Thus, the SPA system reduces spam by stopping it right before the user sees it. Encoding policy as part of the email address allows for

stateless incoming mail filtering. There is no need to manage whitelists, and blacklists, to the extent that they will be needed, can have addresses removed after their expiration date. Integrity checks protect the SPA from tampering, and encryption ensures that adversaries cannot infer valid, non-SPA addresses from the SPA itself; in a limited sense, SPA therefore also reduces exposure of email addresses. Finally, while it is better to get some support from the mail server, the SPA system can work with reduced functionality with just client support, increasing its chances of adoption.

Acknowledgments

Many thanks go to Viktor Dukhovni for his help with *Postfix*, and to the anonymous reviewers for their helpful comments.

References

[1] `http://www.spamlaws.com/`.

[2] `http://www.tmda.net/`.

[3] `http://www.chooseyourmail.com/`.

[4] `http://www.spamassassin.com/`.

[5] A. Back. Hashcash — A Denial of Service Countermeasure. `http://www.cypherspace.org/hashcash/`, April 1997.

[6] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The KeyNote Trust-Management System Version 2. RFC 2704, `http://www.rfc-editor.org/`, September 1999.

[7] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proc. of the 17th Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, Los Alamitos, 1996.

[8] M. Crispin. Internet Message Access Protocol — Version 4rev1. RFC 2060, `http://www.rfc-editor.org/`, December 1996.

[9] D. H. Crocker. Standard For The Format Of ARPA Internet Text Messages. Request for Comments 822, Internet Engineering Task Force, August 1982.

[10] J Klensin, Editor. Simple Mail Transfer Protocol. Request for Comments 2821, Internet Engineering Task Force, April 2001.

[11] M. Blaze and J. Feigenbaum and J. Ioannidis and A. Keromytis. The Role of Trust Management in Distributed Systems Security. In *Secure Internet Programming*, pages 185–210.

[12] S. H. Mueller. Spam and the Law. *;Login:*, April 1998.

[13] J. Myers and M. Rose. Post Office Protocol — Version 3. RFC 1939, `http://www.rfc-editor.org/`, May 1996.

[14] J. Postel. Simple Mail Transfer Protocol. Request for Comments 821, Internet Engineering Task Force, August 1982.

[15] Wietse Venema. `http://www.postfix.org/`.