Bounce address protection for email

Tony Finch <fanf2@cam.ac.uk> <dot@dotat.at>

University of Cambridge Computing Service

ABSTRACT

One of the more vexing problems with the lack of security in Internet email is "collateral spam": the backscatter of bounces from forged email that is sent to the forged address. The effect of this is for users to ignore bounces as just another kind of spam, and even for some email system administrators to disable bounces altogether. So as well as being a nuisance, it is making email less reliable.

This paper will describe a technique for determining whether a bounce is in response to a legitimate message or if it is collateral spam. I'll describe the difficulties which make the technique more complicated to implement and deploy, and my approach to a solution for Cambridge University. I'll also describe how the technique can be extended to detecting the original forgeries as well as the backscatter.

1. Introduction

Internet email [1, 2] has no transport-level protection against forgery. By "forgery" I mean lying about the origin of a message; it does not necessarily imply any sophisticated craftsmanship to hide the lie. There have for a long time been tools such as PGP [3] for securing email message data – the user-level part – however these do not protect the transport-level mechanisms of email, such as reporting delivery failures.

Email is not completely lacking in security: there are specifications for SMTP authentication [4] and encrypted SMTP [5]. However these are aimed at securing the current SMTP connection between the client and the server, so they authenticate the most recent client which is usually different from the message's original sender. They have poor scaling properties for the management of authentication credentials – $O(|clients| \times |servers|)$ – and they require prior arrangement before email can be exchanged. This means that while they are suitable for securing the submission of messages by users to their email provider's outgoing servers, they are not capable of authenticating message transfer across the public Internet.

Delivery failures and delays are handled by sending a "delivery status notification" – colloquially known as a bounce message – to the original message's "return-path". As in postal snail-mail, an email message has an envelope bearing a return address (which usually refers to the message's sender) and one (or more) recipient address(es). The message data is divided into a header and a body. As well as miscellanea like the date and subject, the header contains addressing information for reference by users, since this is not necessarily the same as that in the envelope. For example if the message was "blind carbon-copied" the envelope will bear recipient addresses not mentioned in the header, or if the message was automatically forwarded the envelope will bear the forwarding address but the header will have the original address.

The fact that delivery failures cause a new message to be sent interacts particularly badly with forgery. If the recipient address turns out to be invalid, the innocent victim of the forgery will be sent a bounce message – i.e. collateral spam, by analogy with collateral damage. Non-bounce auto-responses such as vacation messages cause similar problems. When a user's address is used in a large spam-run (known as a "joe job" after the victim of an early incident) the volume of backscattered collateral spam can be overwhelming. Continuous backscatter is troublesome even in small quantities because it can lead to users ignoring all bounces since from the user's point of view they are usually meaningless technical mumbo-jumbo. What is worse is that some system administrators and software developers [6] are causing all bounces to be rejected as some kind of misguided anti-spam measure. This completely wrecks email's error handling, causing messages to disappear without trace.

This paper was presented at the UKUUG LISA Winter Conference on Security & Networks held on the 24th & 25th February 2005 in Birmingham. It is published in the conference proceedings and on the Web at `http://www.cus.cam.ac.uk/~fanf2/hermes/doc/talks/2005-02-ukuug/`

c: MAIL FROM:<*return-path*>
s: **250 OK**
c: RCPT TO:<*recipient*>
s: **250 OK**
c: RCPT TO:<*bcc recipient*>
s: **250 OK**
c: DATA
s: **354 Enter message, ending with '.' on a line by itself**
c: From: *author*
c: To: *recipient*
c: Subject: example
c:
c: *message body*
c: .
s: **250 OK id=1CzDnP-0002AX-1c**

Fig. 1: A message being sent via SMTP. Lines sent by the client are marked "c:" and lines sent by the server are marked "s:". The envelope comprises the MAIL FROM and RCPT TO commands. The message header follows the server's response to the DATA command, up to the first blank line. This is followed by the message body, up to the terminating dot.

What we want is a means to distinguish between bounces arising from legitimate messages and collateral spam from forgeries, in order to remove the irritation and restore the integrity of email's error handling. It is possible, to some extent, to use heuristics to do this; however this approach cannot correctly handle certain kinds of auto-response described in the next section. The technique described by this paper uses cryptography: a unique unforgeable security tag is added to all outgoing messages, and all bounces are checked to see if they contain a valid tag. If not, the bounce is collateral spam and can be rejected. This requires no co-operation from other sites on the Internet, so it can be implemented unilaterally and without standardization. However, if the mechanism is extended to allow recipients to verify the tag, then it can be used as a general anti-forgery system, not limited to just bounces.

Other people are working on similar technology and some have deployed prototypes. David Woodhouse [7] has been using his implementation for over a year. A group led by John Levine and Dave Crocker [8] is aiming to produce an IETF specification. The scheme being developed by Seth Goodman, James Cousins, and others [9] includes message data digests and call-back verification features. This paper describes the work in progress at the University of Cambridge [10] which will be deployed on a system with over 30 000 users. It is known as "DCBA", for (email) domain cryptographic bounce authentication.

The next section describes where our scheme places the security tag, and the reasons for that placement. Section 3 describes the syntax of the tag and the cryptography used to generate and verify it. Section 4 examines potential interoperability problems and how these are addressed. Section 5 discusses how it can be used as a general anti-forgery mechanism. Section 6 is concerned with the weaknesses in its security. Finally, Section 7 contains some concluding remarks.

2. Placement of the security tag

It is not immediately obvious where to put the tag so that it is reliably present and verifiable in bounces and auto-responses. Other anti-forgery systems have placed their signatures in the message body (for example, the q249 scheme [11]) or, more commonly, in the message header (as is done by Yahoo! DomainKeys [12] and Cisco IIM [13]). However it is quite common for bounces to omit the body of the original message, or to abbreviate its header in a way that leaves out any signature fields. In addition to that, certain kinds of auto-responses, such as vacation messages, look like bounces at the transport level but do not include any of the original message data. This also means that it isn't possible to use anti-spam heuristics to identify collateral spam reliably.

We observe that the distinguishing feature of bounces and many other kinds of auto-response [14] is that they are sent to the return path of the original message, and they have no return path of their own.[1] Therefore we put our security tag in the return path of any message we send. When receiving a message with a null return path, we check that the recipient address has a valid tag; if it has no tag or an invalid tag, we reject the message.

[1] Thus loops cannot occur, because a bounce that cannot be delivered does not have a return path to send the failure report to; instead it is brought to the local postmaster's attention.

```
c: MAIL FROM:<>
s: 250 OK
c: RCPT TO:<return-path>
s: 250 OK
c: DATA
s: 354 Enter message, ending with '.' on a line by itself
c: From: Mail Delivery System <Mailer-Daemon@recipient-domain>
c: To: return-path
c: Subject: Mail delivery failed: returning message to sender
c:
c: bounce rubric
c:
c: ------ This is a copy of the message, including all the headers. ------
c: From: author
c: To: recipient
c: Subject: example
c:
c: original message body
c: .
s: 250 OK id=1CzE3a-000418-0R
```

Fig. 2: A bounce being sent by SMTP. Bounces are distinguished by having a null return path, indicated by <>, and they are sent to the return path of the message that could not be delivered. The bounce rubric explains the reason why the original message could not be delivered.

Apart from being the only approach that is likely to work, this mechanism has some further advantages. It only depends on the envelope of the bounce message, which means we can make the rejection decision before the bounce's data has been transmitted, saving bandwidth. We don't have to parse the bounce data, just an email address. Further advantages apply when the mechanism is extended to allow recipients to detect message forgery, as discussed in section 5.

We still need to decide where in the return path the tag should be placed. There are two possibilities: either in the local part of the address (before the @) or in the domain part. The local part is the more natural place: it is only handled by email software, so only email software has to be modified to support the new functionality. Both SES and BATV [9, 8] put their tags in the local part. However, the local part does not have very much space: RFC 2821 [1] only guarantees 64 characters, and although the specification provides a fairly generous alphabet some implementations are upset by odd punctuation or smash the case of letters.

In our implementation I have decided to put the tag in the domain part of the return path. The extra space – typically 200 characters – allows wider scope in the choice of cryptography, especially because public key signatures tend to be large. This becomes more relevant when we consider third-party verification in section 5. Another advantage is that it provides more possible defences against the weaknesses of the scheme, as I will discuss in section 6. Of course the big disadvantage is that it means the DCBA implementation must hook into the DNS as well as the email software, which makes deployment more complicated.

3. Syntax and cryptography of the security tag

The security tag is a signed email address. Its syntax is based on a generic framework which supports different types of signature. For example, the signed version of `fanf9@hermes.cam.ac.uk` has the form:

`fanf9@`*sig.nat.ure.type*`.fanf9.a--t.hermes.cam.ac.uk`

The `a--t` part is a marker indicating to recipients that this is a DCBA address (as discussed further in sections 4 and 5).[2] The local part is duplicated so that a signature covering the whole address can be verified given just the domain part. (This is discussed further in section 5.) Characters that are valid in a local part but not a domain are encoded; the details are omitted here for space reasons. The *type* part is a keyword which indicates the format of the signature. The signature itself can consist of more than one component of the domain, so that it is not restricted to a 63 character limit.

We define two standard signature types, one for general use, and a special-case one for interoperability purposes discussed in the next section. They are not signatures in the public-key crypto sense, although public-

[2] It's slightly surprising that double hyphens are permitted in domain names: see section 4.2.1 of [15]. Internationalized domain names use them in the punycode label prefix `xn--` to make non-IDN collisions unlikely.

key signatures could be used in the future. I only give a rough overview of their generation and verification here; a full specification will be published on the web [10].

The `hmac-sha1` signature type has three parts: *mac*`.`*time*`.`*nonce*; i.e. a nonce, an expiry time, and a message authentication code [16, 17]. A "nonce" is random meaningless padding which provides additional variability to the data covered by the MAC. For our purposes the nonce must be fixed for a given message (as explained in section 4), so it could be based on the message-ID or a hash of part of the message data. The expiry time provides some defence against the attacks described in section 6. The MAC is generated using a secret key and the rest of the address. Verification requires knowledge of the secret: it involves checking that the address has not expired, then re-generating the MAC and comparing it with the one in the address; no special knowledge about the construction of the nonce is required.

The `fixed` signature type has two parts: *key*`.`*nonce*; i.e. a lookup key and a nonce. Unlike `hmac-sha1`, it does not vary per message nor does it have an expiry time. The lookup key is derived from a hash of the message's sender and recipient addresses, and is used to retrieve the nonce from a database. A copy of the database is required to verify addresses with `fixed` signatures; the database entry includes enough information to verify the rest of the address, and no special knowledge about the construction of the key is required. If the key is not present in the database a default nonce is generated algorithmically using the HMAC of a secret, the lookup key, and the rest of the address. This reduces the size of the database and simplifies the provisioning of new accounts. The effect of this optimization is that the database contains only non-default nonces resulting from revocation of default `fixed` signatures (see section 6).

4. Interoperability

Forgery protection schemes impose some new requirements and rely on assumptions which may conflict with other email systems. This section examines some ways in which this can cause problems and how they are mitigated. Although I often refer to DCBA, the problems apply to any similar scheme [7, 8, 9].

4.1. Consistent tagging

Forgery protection relies on legitimate messages being consistently tagged, otherwise they will be falsely identified as forgeries. If DCBA is only being used to protect against collateral spam, this problem merely leads to unreliability; however if it is extended to general forgery protection then an untagged message will be silently lost, because both it and its bounce will be rejected.

Consistent tagging basically requires that all messages are sent via the email provider's servers, since only they can add valid tags. However in the past roaming users and people working from home had to use their connectivity provider's smart host. Before DCBA can be deployed they must update their configurations to the modern recommendation, so that they always use their email provider's secure message submission service [4, 5, 18, 19]. However this is difficult to achieve across a large and mostly uninterested user base.

The alternative approach is to make DCBA a per-user option. Security tags are only added to messages submitted by authenticated opted-in users, and verification is only applied to messages purportedly from opted-in users or bounces destined to them. As well as making deployment easier, this means that the security tag is tied to a particular user; a valid tag then implies not only that the message is legitimate but also that the message's sender information can be trusted. This is a much stronger guarantee than is required for protection against collateral spam, or for authenticating that a message came from a particular organization: it makes it possible to protect against forgery between users within an organization.

4.2. Assumptions about the return path

The original SMTP specification [15] stated that the argument to the `MAIL FROM` command is the email address of the message's sender. However from the protocol point of view its function is to identify where auto-responses should be sent, which is not necessarily the same address. For example, it is common for mailing list management software to have a special bounce handling component; its address is used as the return path of a message sent to a list, and it is different from the list's address that is placed in the message's `Sender:` header field. Hence "bounce address" is an increasingly popular alternative name for the return path.

However it is not uncommon for software to use the return path to identify a message's sender, and to assume that a given sender will use a consistent return path. DCBA breaks this assumption because each message has a different return path. This causes serious problems for some mailing list software, and irritating problems for some anti-spam systems. Fortunately the same remedies work for all of them.

Mailing list management software such as ezmlm [20] which identifies list subscribers by their return paths is incompatible with DCBA. Users will not be able to subscribe or unsubscribe because their request will appear to come from an inconsistent ‘‘sender’’ (though users’ existing subscriptions will continue to work after they turn on forgery protection); if the mailing list restricts posting to subscribers, users will not be able to post because their return path will not match their subscription address.

Greylisting [21] is an anti-spam technique based on the observation that common spamming software forges email promiscuously, and has a minimal SMTP implementation that does not retry after temporary delivery failures. A greylisting server maintains a database of triples (IP address, return path, recipient address) and temporarily rejects messages with triples that have not been seen before. If the client tries to send the message again within a reasonable period of time it is accepted and its triple is added to the database. Subsequent messages with the same triple are accepted without delay. Every message sent with DCBA has a different triple so will always be delayed by a greylisting recipient. We stated in section 3 that the security tag must be fixed for a given message; this is partly so that DCBA messages aren’t perpetually delayed by greylisting sites because they never have the same triple.

Some anti-spam systems maintain a whitelist of known senders and send a challenge to unknown senders. If the challenge is correctly answered the sender is added to the whitelist. If the return path is used to identify the sender, and the sender uses a different return path for every message, then the sender will get a challenge for every message. This makes challenge/response systems even more irritating.

There are three solutions to these problems. The best solution is to fix the problem software to use the `Sender:` header field to identify a message’s sender. The next best (but at least as unlikely to succeed) requires a standard format for signed bounce addresses – such as the generic framework defined in section 3 – which the problem software can recognize and thereby recover the unsigned version of the sender’s address. More realistically, DCBA can be adapted to use a fixed return path so that it does not provoke the problems. Doing so globally would be equivalent to not using DCBA, so we only do it selectively.

We maintain a list of problem recipients, and when a message is sent to one of them it is tagged using a `fixed` signature. This is only constant for a given recipient, which means that if the tag is leaked and used to forge email, the responsible recipient can be identified.[3] The leaked address can be revoked in order to stop further collateral spam, though this means that the user will have to re-subscribe to the mailing list etc. An alternative approach, which is more user-friendly though at significant cost in reliability, would be not to tag messages to selected problem recipients, and hope that none of them bounce.

4.3. Non-conformant auto-responses

DCBA assumes that bounces are well-formed. This assumption can be broken in two ways: by bounce messages that do not have null return paths; and by bounce messages that are not sent to the original message’s return path.

A fairly large proportion of collateral spam is caused by idiot anti-virus software that sends a warning to the ‘‘sender’’ of an infected message (despite the fact that most infected messages are forgeries). These auto-responses are often sent with a return path that refers to the anti-virus software, so they are not rejected by DCBA. However standard anti-spam pattern-matching techniques are effective at eliminating these messages. Some vacation auto-responders have a similar problem, which can be similarly dealt with. However this may cause legitimate vacation messages to be lost.

The other kind of problem arises when auto-responses that have a null return path are sent to the address in the `Reply-To:` header field of the original message, to pick a common mistake. These kinds of auto-responses are more worrying because of the likelihood that a desired message will be lost because it looks like collateral spam. It is hard at the moment to asses how widespread this kind of auto-responder is, but we hope that experience with using DCBA will show they are rare, and that the recommendations for auto-responders in RFC 3834 [14] may help us to get broken ones fixed.

5. Verification of non-bounce messages

Detecting collateral spam involves verifying the security tag in bounce messages; if normal messages are verified too then DCBA provides general forgery protection. This is useful even if DCBA is only deployed in one place, because it can deal with the relatively common forgeries where an organization receives a message from outside that purports to come from inside. A large site has a user population that is more likely to include untrustworthy people, so it will benefit from the stronger internal protection. DCBA works well in places with complicated email routeing because messages can be verified at any point on their journey.

---

[3] This does not necessarily imply that the recipient is criminal; they might legitimately publish email in a way that exposes tags to spammers. See section 6 for examples.

```
c: MAIL FROM:<>
s: 250 OK
c: RCPT TO:<return-path>
s: 250 OK
c: RSET
s: 250 Reset OK
```

Fig. 3: An SMTP call-back is an abbreviated protocol exchange consisting of just a bounce message envelope, which is used to verify a return path address.

However significant advantages accrue if tags can be verified by other sites across the public Internet. They are then protected from forged spam and virus-bearing email that claims to be sent by someone who uses DCBA. If the purported sender of a message is reliably believable then it can be used by reputation systems such as blacklists. The reason that existing blacklists are based on IP addresses is that IP addresses are currently the only reliable identity in email. However they are very low-level, so there is a significant desire for anti-spam systems based on more user-friendly identities.

5.1. SMTP call-backs

Some modern SMTP software supports "call-out" remote address verification in order to detect more incorrectly addressed email. There are two kinds of call-out: a "call-forward" verifies a recipient address and a "call-back" verifies a return path. They work by performing an abbreviated SMTP protocol exchange consisting of just a message envelope; the exchange is reset before the message data is transferred.[4] In call-forward verification the SMTP conversation proceeds as if the message was being relayed on to its destination, with the same return path and recipient addresses. In call-back verification it is as if the message was being bounced: the return path is null and the recipient address is the return path of the original message.

If the sending site uses DCBA it will verify the address in the envelope of a bounce message as part of its collateral spam detection. Its response will be positive only if the address has a valid signature. Thus the result of a call-back to this site will be positive for legitimate email and negative for a forgery. Recipient sites that use call-back verification are automatically protected from forgeries that purport to come from a site that uses DCBA, without any special knowledge of DCBA!

5.2. DNS call-backs

Most sites do not implement SMTP call-back verification. However it is relatively common for them to implement a weaker kind of remote address verification that only checks the domain part: they look up the domain in the DNS and check that it has a plausible MX or A record. DCBA is designed to work with this mechanism. When a site receives a message with a signed address it will make a query containing the security tag (i.e. the domain) to the sending site's DNS server; as mentioned in section 3, this query has enough information to reconstruct the whole address. The DNS server can therefore verify the DCBA address and return a positive or negative response accordingly. (This requires a "stunt" DNS server which has application-specific logic in addition to a DNS protocol implementation.) It can also implement the rate-limiting defence described in section 6.

Although this design is very powerful, it has a large gap: it does not protect against forged messages that use unsigned addresses in the return path, only against old or abused signed addresses. Therefore it misses most forgeries. In order to fix this we need co-operation from recipients: they must check whether a return path should be signed. When a message with an unsigned return path arrives, the recipient can perform a DNS lookup for a record under the return path's domain. If the result indicates that the return path should have been signed then the message can be rejected. I mentioned in section 4 that we will make DCBA a per-user option, so this DNS lookup will have to include the local part of the address as well as its domain. We can use section 3's generic framework to construct an appropriate DNS name to look up, for example:

```
options.fanf9.a--t.hermes.cam.ac.uk
```

DNS call-backs also require a complicated implementation with a modified DNS server as well as a modified SMTP server. Fortunately DCBA can be used with a standard DNS server: a suitable wildcard record will cause it to give a positive response to any address that appears to be signed, whether valid or not. This allows

[4] Although SMTP has a `VRFY` command designed for performing remote address verification, it is not used for call-outs because it has for a long time been standard practice is to disable it. For our purposes it has the additional disadvantage of being unable to distinguish between forward and reverse verification.

prototypes to be developed in stages.

5.3. Call-back scalability

There are a number of concerns about the scalability of call-backs. SMTP call-backs are particularly heavy-weight, partly because they use TCP and partly because of the design of typical SMTP servers and their burden of anti-spam checks. Widespread implementation of call-backs would make the effect of a joe job rather worse than it is now, since the victim site would have to deal with the call-back load as well as the collateral spam. Call-back implementations – both SMTP and DNS – maintain caches to avoid redundant queries. However DCBA deliberately ensures that most call-backs will be checking different addresses, which makes caches ineffective and bloated with useless entries.

For an Internet-scale protocol, we would like an architecture which spreads the load of distributed attacks and which makes good use of caches. DCBA is currently designed for unilateral deployment, so it tries to maximize the strength of its defences in the absence of DCBA implementations at other sites, and this limits its scalability. However its design does not rule out protocols which make more effective use of co-operation between sites. We will discuss some options in the next sub-section.

5.4. Public key cryptography

If DCBA is augmented with a signature type based on public key cryptography, then recipients will be able to verify security tags themselves without calling back to the sender. Keys are also much more cacheable than signatures. This seems to address our scalability problems nicely, but it introduces some new problems of its own.

Public key signatures tend to be quite large. DCBA has about one kilobit of space for a signature (200 characters of 5 bits each) which might be enough room for an elliptic curve signature but is not enough for RSA. This space restriction makes the implementation more challenging.

A common problem with public key cryptosystems is key revocation. How can we stop a signed address from being used for spamming? We can't revoke a key that is used for the whole domain since it will affect all email in transit. If we have per-user keys we can target revocation more precisely, however this is only acceptable if the abuse is caused by the user. Therefore, as discussed in section 6, the security tag must be tied to the message data so that spammers can't replace a legitimate message with their own. An alternative to per-user keys is a mechanism for advertising which accounts are valid; either can be implemented as an extension to the `options` lookup described earlier.

DomainKeys and IIM [12, 13] both use public key signatures. DomainKeys allows per-domain or per-user keys, and deals with abuse by key revocation. IIM also has a choice of keying models, but has a separate call-back service for checking who is authorized to use a key. Since they both put signatures in the message header, they do not have the same space worries as a public key version of DCBA. However they cannot provide protection from collateral spam, so it seems sensible to view them as complementary technologies rather than competing. Another reason for this view is described towards the end of the next section.

6. Weaknesses and defences

So far we have not examined DCBA in detail from the point of view of an attacker. This section describes some security weaknesses of DCBA and related forgery protection schemes, and how the basic idea can be strengthened to protect against them. I'm treating DCBA as a general forgery protection system for the purpose of this analysis, because collateral spam is just a special case of forgery, and the wider problem affords more defence mechanisms.

6.1. Security tag exposure

If Professor Vladimir Important sends a message to his undergraduate student Amy Miscreant, then A. Miscreant can use the security tag in the message to send forged V. Important messages. Although we have adequate ways of dealing with pranks like this, it is more worrying if criminals can get hold of security tags and use them for spamming. Unfortunately there are a number of ways they can do this.

Messages may be included in public web archives (for example, bug tracking systems) from which signed addresses may be extracted. Another common source of addresses used for forgeries is computers that have been hijacked by a virus or other malware. "Promiscuous senders" such as help-desk staff can easily be provoked into sending a tagged message to a spammer. Criminals can get an account with an ISP, send a message to themselves in order to obtain a valid security tag, then use that for spamming.

It's worth noting that mailing lists – which frequently involve web archives and promiscuous distribution of messages – are not much of a problem. They usually replace the return path of messages they re-send with the address of the mailing list manager's bounce handler, so the security tag on the original message is lost. On the other hand this benefit is easily negated if the message happens to go through a system that records the return path in the message's `Received:` trace fields.

6.2. Revoking security tags

The most basic security measure is to limit the lifetime of a signed address, hence the expiry time in the `hmac-sha1` signature type. However the tag must be valid for at least as long as it takes to deliver the message, and although this is usually a matter of seconds it may take days. Tag lifetimes must be at least as long as common MTA retry times, which are often a week and may be as much as a month. Despite being weak, this technique is effective at limiting the usefulness of old email archives for an attacker.

One can also revoke a tag early if it is abused. This is the approach taken for the `fixed` signature type, though it can be applied to any type. However in many cases this will be closing the stable door after the horse has bolted. Detecting abuse early enough to make revocation useful is very difficult because email provides no feedback about successful deliveries, and even if feedback is available (see section 5) it is very difficult to distinguish between a legitimate mail-shot and abuse. It might seem reasonable to scale the abuse threshold according to the number of recipients of the message; however this will not work if one of the recipients is a mailing list reflector that leaves the return path intact.

6.3. Rate-limiting

Rather than drawing a hard line between valid and invalid, a more flexible approach is to limit the rate at which copies of a message may be delivered. In order for this to be possible a call-back mechanism is required, such as DCBA's DNS call-backs described in section 5. Verification requests that come too quickly cause a temporary failure, for example an SMTP `4XX` response or a DNS `ServFail` response, so that message delivery is deferred and retried later.[5] The maximum rate can be made a function of the age of the security tag: young tags may be verified at a high rate, allowing messages to be delivered immediately, but the permitted rate drops off fast so that spammers are unlikely to have time to exploit it effectively. The rate remains non-zero until the tag expires.

Note that this is a per-message rate limit, so it doesn't limit a user's overall volume of email – though this is also a good idea, to protect against hijacking of the user's computer. High-volume email should probably be sent via special mechanisms such as mailing list software. Therefore this defence should not inconvenience legitimate email, but it may be enough to prevent or mitigate the criminal ISP customer attack.

6.4. Message data signatures

The professorial spoofing that I mentioned earlier is possible because there is no explicit link between the security tag and the message it is attached to. Attackers can re-use a signed address with any message of their choosing. This class of attack can be prevented if the signature in the security tag also covers the message data. For example, the nonce in the `hmac-sha1` signature could be a digest of the message data rather than an arbitrary value. The exact details of the digest are difficult to define because of in-transit changes to the message data, especially the header. A good approach would be to allow DCBA to delegate this job to a protocol designed for the purpose, such as DomainKeys or IIM [12, 13], and put a hash of the DK or IIM signature in the DCBA signature.

This defence is attractive but has a number of limitations. Full signature checking can only be performed by the message recipient, because only the recipient has the full message data: call-back checks only include the digest, and bounce messages are usually too mangled for reliable checking. Therefore this defence depends on widespread deployment to be effective. Also, it only protects against message modification attacks, and does not prevent spamming with validly-signed messages, so it's a useful addition to the other defences described in this section rather than a replacement.

7. Conclusion

At the moment DCBA is still a work in progress, so it is still too early to draw any conclusions. We have put some significant infrastructure in place at Cambridge which is necessary to support DCBA, in particular authenticated message submission. I have developed prototype changes to our Exim configuration, which implement the logic for tagging newly-submitted messages and verifying the tags on bounces and other

[5] This may cause a similar effect to greylisting in the event of a spam attack.

messages, on a per-user basis. The Exim configuration calls separate code that implements the details of signing and verifying addresses. This code has not been completed yet.

In this paper I have examined various considerations behind the design of DCBA, in particular those related to deployability, interoperability, and security. I have touched on its relationship to DomainKeys and IIM [12, 13] and how it might provide a useful adjunct to them. There is a lot of work taking place at the moment on standards for email security. We hope that experience using DCBA in a large organization will provide useful input to this process.

References

1. John C. Klensin (ed), *Simple Mail Transfer Protocol,* RFC 2821 (Apr 2001). `http://www.ietf.org/rfc/rfc2821.txt`
2. Peter W. Resnick (ed), *Internet Message Format,* RFC 2822 (Apr 2001). `http://www.ietf.org/rfc/rfc2822.txt`
3. Jon Callas, Lutz Donnerhacke, Hal Finney, and Rodney Thayer, *OpenPGP Message Format,* RFC 2440 (Nov 1998). `http://www.ietf.org/rfc/rfc2440.txt`
4. John G. Myers, *SMTP Service Extension for Authentication,* RFC 2554 (Mar 1999). `http://www.ietf.org/rfc/rfc2554.txt`
5. Paul Hoffman, *SMTP Service Extension for Secure SMTP over Transport Layer Security,* RFC 3207 (Feb 2002). `http://www.ietf.org/rfc/rfc3207.txt`
6. Ipswitch, Inc., *IMail - 501 bogus mail from in SMTP log.* `http://support.ipswitch.com/kb/IM-19980116-DD02.htm`
7. David Woodhouse, *Reverse-Path Rewriting (aka Sender Rewriting Scheme) in Exim 4* (Sep 2004). `http://www.infradead.org/rpr.html`
8. John Levine, Dave Crocker, Sam Silberman, and Tony Finch, *Bounce Address Tag Validation (BATV),* Internet Draft (Sep 2004). `http://www.ietf.org/internet-drafts/draft-levine-mass-batv-00.txt`
9. Seth Goodman and James Couzens, *The Signed Envelope Sender (SES) Protocol.* `http://ses.codeshare.ca/`
10. Tony Finch, *Anti-forgery project documents.* `http://www.cus.cam.ac.uk/~fanf2/hermes/doc/antiforgery/`
11. Russell Nelson, *q249* (Jan 2003). `http://web.archive.org/web/q249.org/`
12. Mark Delany (Yahoo! Inc.), *Domain-based Email Authentication Using Public-Keys Advertised in the DNS (DomainKeys),* Internet Draft (Aug 2004). `http://www.ietf.org/internet-drafts/draft-delany-domainkeys-base-01.txt`
13. Jim Fenton and Michael Thomas (Cisco, Inc.), *Identified Internet Mail,* Internet Draft (Oct 2004). `http://www.ietf.org/internet-drafts/draft-fenton-identified-mail-01.txt`
14. Keith Moore, *Recommendations for Automatic Responses to Electronic Mail,* RFC 3834 (Aug 2004). `http://www.ietf.org/rfc/rfc3834.txt`
15. Jonathan B. Postel, *Simple Mail Transfer Protocol,* RFC 821 (Aug 1982). `http://www.ietf.org/rfc/rfc821.txt`
16. Hugo Krawczyk, Mihir Bellare, and Ran Canetti, *HMAC: Keyed-Hashing for Message Authentication,* RFC 2104 (Feb 1997). `http://www.ietf.org/rfc/rfc2104.txt`
17. Donald E. Eastlake, 3rd and Paul E. Jones, *US Secure Hash Algorithm 1 (SHA1),* RFC 3174 (Sep 2001). `http://www.ietf.org/rfc/rfc3174.txt`
18. Randall Gellens and John C. Klensin, *Message Submission,* RFC 2476 (Dec 1998). `http://www.ietf.org/rfc/rfc2476.txt`
19. University of Cambridge Computing Service, *Mail program settings for Hermes.* `http://www.cam.ac.uk/cs/email/muasettings.html`
20. Daniel J. Bernstien, *ezmlm.* `http://cr.yp.to/ezmlm.html`
21. Bjarne Lundgren, *Greylisting: a great weapon against spammers.* `http://www.greylisting.org/`