Network Working Group
Internet-Draft
Intended status: Experimental
Expires: 17 September 2026

A. Vesely

16 March 2026

Agreements To Fix Forwarding
draft-vesely-fix-forwarding-06

Abstract

The widespread adoption of Domain-based Message Authentication, Reporting, and Conformance (DMARC) causes problems to indirect mail flows. This document proposes a protocol to establish forwarding agreements whereby a mailbox provider can whitelist indirect mail flows directed toward its users by maintaining per-user lists of agreements.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 17 September 2026.

Table of Contents

1. Introduction

The ability to send messages to anyone without prior agreement is a feature at the base of the success of Internet mail. It also exposes a way to abuse, though. Domain-based email authentication can limit abusive behavior by enabling receivers to unambiguously attribute responsibility of messages, and thereby to develop domain reputation. However, indirect mail flows, such as mailing lists and aliases are seriously hindered by such authentication practices (see [RFC7960]). Although they constitute a minor part of the global mail traffic, such hindrance poses a problem, which is what the present protocol attempts to fix.

Many mailing lists break DomainKeys Identified Mail (DKIM) signatures ([RFC6376]) by adding a subject tag or a message footer. External aliases break Sender Policy Framework (SPF) ([RFC7208]) if forwarders don't change the bounce address, or break its alignment with the From: header field if they do. In both cases, they break Domain-based Message Authentication, Reporting, and Conformance (DMARC) ([RFC7489]) if there is no DKIM signature.

Rewriting the very From: field, an operation known as From: munging, is a sender side solution to this problem, adopted by most mailing lists; see Section 4.1.3.1 of [RFC7960] for a detailed description. Alias forwarding, instead, can be avoided by having the target pull messages using a mail retrieval protocol rather than pushing them via SMTP. As an alternative, this document proposes a cooperative method that hinges on the peculiarity that both of these forwarding methods require a prior setup at the sender. At that stage, it does not cause impediment to also set up an agreement with the receiver.

1.1. Presentation by example

Let's consider the procedure whereby alice@example.com subscribes to the mailing list participants@lists.example.org. The first three steps below are a well known practice known as *confirmed opt-in* (COI). The extra steps are introduced by this protocol:

1. Alice subscribes to the list, either by visiting example.org web site or by sending a request to the list manager.

2. The mailing list manager (MLM) sends a message to Alice, asking whether she agrees to subscribe.

3. Alice confirms the subscription.

4. The MLM looks up _fixforwarding.example.com on the DNS to check if Alice's provider supports forwarding agreements (see Section 4 for the DNS record format). If found, the MLM applies for a permission to send mailing list messages that may fail DMARC checks (see Section 5 for the web form fields).

5. Upon receiving the MLM request, example.com sends a message to its user Alice, asking whether she agrees to receive that specific mail flow.

6. Alice confirms to her provider too. (This step may eventally replace step 3 above.)

7. Example.com adds participants@lists.example.org to the list of forwarders that Alice agreed upon. Then, it confirms to the MLM that the agreement is set up.

8. The MLM modifies Alice’s subscription options removing From: munging for messages sent to her.

Please note that these are very simple steps, requiring very simple software. Steps 2 and 3 may be removed when MLMs trust that steps 5 and 6 work well. The protocol is designed in such a way that it can be adopted by large and tiny providers alike.

Step 7 constitutes the actual setup of the agreement on the receiver side. It enables example.com to recognize messages that belong to this mail flow by:

i. authenticating example.org by verifying its ARC- or DKIM-signature, and

ii. checking that the authenticated List-ID: header field is included in the list of forwarders that Alice agreed upon. This requires the DMARC verifier to be able to perform per-user exemptions; possibly the only relevant software requirement.

According to the user’s will, DMARC failures can be safely ignored under these conditions.

Step 8 constitutes the protocol’s payload. It requires the MLM to be able to configure From: munging on a per-subscriber basis; Appendix A proposes a method to achieve such effect.

These steps are discussed in detail in the following sections.

2. Terms Definitions

forwarder
The entity that routinely forwards messages it has received to one or more external recipients. Forwarders are roughly divided into two categories, aliases and mailing lists, much like SMTP definition ([RFC5321], Section 3.1).

alias
A way of forwarding to a fixed set of recipients. The bounce address may or may not be changed.

mailing list
A mechanism whereby subscribers can dynamically add or remove themselves from the list. The bounce address is always changed.

bounce address
Also called envelope return address, is the parameter of the SMTP MAIL command.

receiving domain
Is the domain, more precisely its MX server(s), which receive the messages sent by the relevant forwarder.

Signers and *verifiers* are defined by DKIM ([RFC6376]).

The use of the term *Mailing List Manager*, almost always abbreviated *MLM* follows [RFC6377]. A MLM is a kind of *Mediator* in [RFC5598] parlance. It is usually composed of a Message Transfer Agent (MTA) with the usual suite of tools plus the mailing list specific software.

Message is defined in [RFC5322]. It consists of a *header* made up of one or more *fields*, and a *body* possibly composed of various MIME *entities*, the latter being defined in [RFC2045] and companions.

We use *colon* (:) to indicate header field names, as in From:, To: and the like.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Note About Rewriting the Bounce Address

This section is about alias forwarding, pinpointing in which cases it is necessary to change the bounce address.

Traditionally, alias expansion re-injects messages in the mail system changing just the envelope recipient address. That is, without rewriting the bounce address. This is what SMTP [RFC5321] prescribes. However, from an authentication perspective, this doesn’t always work, because some MTAs reject SPF failures —those matching a -all directive (see SPF spec, [RFC7208])— during the early stages of the SMTP protocol; that is, before DKIM or ARC [RFC8617] can be verified.

It is handy to leave the bounce address intact in order to have the message author notified in case of failed delivery, which increases email reliability. However, some types of errors, such as a failed delivery because the destination mailbox has been deleted, should be

sent to someone who deletes the forwarding configuration. A broken alias which unflaggingly generates bounces to any bounce address is akin to an open relay. Albeit limited to non-delivery notifications, it can be exploited for spamming. To avoid this role, a receiver can carefully verify the bounce address before accepting a message, so a forwarder can expect the receiving domain to behave accordingly.

We recommend that the bounce address be replaced with the internal address of a human or robot that can figure out what to do; that can be done using a sender rewriting scheme ([SRS]). However, this protocol does provide for a way to enter into agreements that cover traditional alias expansion. According to Appendix D of [RFC7208], domains consult a whitelist if they reject SPF "fail" messages before receiving the message data. If the whitelist they consult is a public DNS-based whitelist (DNSWL, [RFC5782]) and if it includes the forwarder’s IP address, then traditional alias expansion, which does not change the bounce address can work even in the face of strict SPF policies.

To recap, an MTA may check SPF after MAIL and before DATA. If the result is "fail", the MTA consults one or more DNSWLs. If the sending IP address is whitelisted, the MTA delays the rejection decision until a later stage, when the message content is received and verified via DKIM or ARC. Therefore, an agreement involving traditional alias expansion is possible if one of the following two conditions is true:

1. The receiving domain’s MXes consult public DNSWLs and the forwarder is subscribed to at least one of them, or

2. the receiving domain’s MXes never reject before receiving and evaluating the message content.

Whether and which of these conditions are met can be determined for each agreement based on the dnswl= tag declaration, as described in Section 4. In cases where none of these conditions are valid, the forwarder MUST change the bounce address.

4. Underscored FixForwarding DNS Resource Record

Receiving domains participating in this protocol notify it to all interested parties by publishing an "underscored" resource record named _fixforwarding. According to [RFC8552], this record is labeled as a subdomain of the domain it refers, which is the domain-part of the pertinent email addresses.

The record contains the conditions that an applicant MUST meet before applying for an agreement as well as a URI to send application requests to. Requests details are specified in Section 5.

The format of the _fixforwarding record follows the extensible "tag-value" syntax for DNS-based key records defined in DKIM [RFC6376]. The folding white space terminal, FWS, is also defined there. The following tags are defined here:

auth
Authentication methods used to identify the forwarder's domain; OPTIONAL, by default the method is ARC ([RFC8617], but a receiving domain MAY state which methods it accepts. A forwarder MUST be able to produce at least one of the required signatures before applying.

The only reason to prefer ARC is because a receiver doesn't know whether the forwarder complies with DMARC policies. ARC provides a way to learn just that.

ABNF:

```
rec-auth-tag        = %s"auth" [FWS] "=" [FWS] rec-auth-tag-method
                      1*( [FWS] "," [FWS] rec-auth-tag-method)
rec-auth-tag-method = %s"arc" / %s"dkim" / future-method
future-method       = 1*( ALPHA / DIGIT / "-" )
```

dnswl
Either a comma-separated list of DNSWLs (see [RFC5782]) that the domain's MXes consult before rejecting SPF failures at the early stages of SMTP transactions, or one of the keywords "none" and "all"; OPTIONAL, by default "none". See Section 3. "none" means this option is not available; that is, forwarders MUST change the bounce address to an SPF valid one. The "all" keyword means the opposite; that is, forwarders MAY leave the original bounce address intact because the receiving domain's MXes never reject messages before verifying their signatures. One or more domains mean the conditional acceptance; that is, if the DNS name that results by prepending the reverse IP address of the forwarder to one of those domains resolves to a positive A IP address (detailed description in [RFC5782]), then the forwarder MAY leave the original bounce address intact because the receiving domain's MXes consult that DNSWL.

ABNF:

```
rec-dnswl-tag      = %s"dnswl" [FWS] "=" [FWS] rec-dnswl-tag-opt
rec-dnswl-tag-opt  = rec-dnswl-tag-list /
                     rec-dnswl-tag-none /
                     rec-dnswl-tag-all
rec-dnswl-tag-list = domain *( [FWS] "," [FWS] domain)
rec-dnswl-tag-none = %s"none"
rec-dnswl-tag-all  = %s"all"
```

Where domain is imported from [RFC5322]. Each domain of the list is a DNS zone that can be queried by prepending reversed IP addresses.

post
This is the URI to post web form requests to. REQUIRED. The web format is described in Section 5. This is an HTTPS or HTTP URI.

ABNF:

```
rec-post-tag     = %s"post" [FWS] "=" [FWS] rec-post-tag-uri
rec-post-tag-uri = https-URI / http-URI
```

Where https-URI and http-URI are imported from [RFC9110].

5. Forwarding Agreement Web Form

Receiving domains participating in this protocol set up a web form through which forwarders can request permission to send messages that may fail DMARC checks. Forwarders wishing to request an agreement with the receiving domain fill out the form providing the required values, mostly email contacts. Since there is a predefined set of fields, applicants can prepare a script which runs automatically when the existence of the form is detected by the _fixforwarding resource record, and the conditions in it are met. However, receiving domains SHOULD provide an HTML form to be filled out manually at the same post= URL, so that an agreement can be entered into even without a dedicated script. When there are no values, the form displays empty fields for filling in; when values are submitted, it displays the result. Values SHALL be submitted using the POST method described in [RFC9110]. Posted values may be encoded using either the application/x-www-form-urlencoded or the multipart/form-data media types (see [RFC7578]).

Normal HTTP rules [RFC9110] apply. In particular, receiving domains SHOULD check that the values posted are acceptable and return a 400 HTTP status code otherwise. If everything is fine, the HTTP server stores the values for later processing and returns status code 202. The resulting agreement status will be sent to the base address of the applicant after checks are completed. Keep in mind that this may take a human lapse of time.

Organizations SHOULD NOT implement access control to restrict access to verified applicants. However, after experiencing an overwhelming number of attacks on the post= URI, an organization capable of designing an interface with a smaller attack surface, can use it to distribute credentials to prospect applicants and thus restrict access to authorized individuals. In this case, the post= URI returns a 401 status code when the applicant is not authenticated; the HTML form page accompanying the 401 code MUST explain how generic operators can obtain the necessary authorization. This process should be automatable once learned, ideally by reusing the same credentials. Requiring authentication is only justifiable for heavily abused domains hosting countless users; otherwise, automated use of this protocol becomes impractical. Forwarders will likely avoid domains that are overly demanding on authorization.

5.1. The Transistor Metaphor

There are three email addresses that play a key role in this protocol. To give them easily memorable names, we use the metaphor of a forwarder being compared to a transistor:

collector
: the entry point where the messages to be forwarded arrive,

emitter
: the target address, where messages are forwarded to, which is the subject of an agreement,

base
: the controlling address, where the state of the agreement is sent, starting with the result of the form submission.

5.2. Form Fields

abuse
: The abuse-team or similar entity email address, where complaints about perceived forwarder’s misbehavior can be sent.

agreement-id

A globally unique string that identifies a request and the related agreement. This has the syntax of a Message-ID field, defined in [RFC5322].

base
The forwarder's email address where the receiving domain sends the agreement acceptance, rejection, renewal or cancellation. These are simple messages readable by both machine and human, described in Messages to the Base.

collector
The mailing list post address, or the address that the alias is attached to.

domain
The forwarder's domain to be authenticated by the receiver. It is the domain indicated in the d= tag of DKIM-Signature: or ARC-Seal: and ARC-Message-Signature: header fields, according to the auth method (Section 4). The domain MUST match the trailing part of the list-id.

emitter
The email address that subscribed to the mailing list, possibly user-confirmed, or the target address of an alias. The domain part of this address MUST be the domain where the _fixforwarding record was found.

list-id
List-ID:s are naturally present in mailing list, and identify the list that the user subscribed to. For aliases, a list-id MUST be synthesized by the forwarder, for example by concatenating an encoded representation of the collector and the signing domain. This field is needed to unequivocally identify the mail flow.

This is the globally unique list-id identifier included in List-Id: header fields in angle brackets as specified by [RFC2919]. Its list-id-namespace MUST be the same domain-name as the domain field of this form. The List-Id: header field SHOULD be signed. Failure to do so can be exploited to damage MLM reputation by replaying suitable messages.

timeout
The number of seconds that the forwarder is committed to wait for a response. The timeout expiring without a result being received is equivalent to a negative result. The timeout SHOULD be greater than 86400 seconds.

text

Free text describing the forwarding reasons or circumstances. This is intended to be presented to the user by the receiving domain when asking for confirmation. It MUST NOT exceed 4K octets, MUST NOT contain HTML tags and MUST NOT contain HTTP or HTTPS URIs.

6. Agreement Management

Agreement data consists of some settings at the receiving domain. Their correspondence to the relevant forwarding settings is handled via Messages to the Base. Agreement data can be created in response to a web form request and removed after cancellation or when the agreement is stale. A web form request can be followed by a rejection response or a timeout, meaning that the agreement was not agreed upon and both parties can forget about it. The forwarder MUST NOT submit the same request again, unless after user intervention.

To honor the agreement, a DMARC filter needs two pieces of agreement data, the emitter and the list-id. It extracts the list-id domain from the message List-ID:. Then, for ARC, the filter checks the validity of the chain and the validity of the last ARC-Message-Signature:, the one with the highest i= tag. If the d= domain of the signature matches the extracted list-id domain, the message might be part of an agreed upon mail flow. For DKIM, a valid signature with d= matching the list-id works in the same way. Once the validity is verified, the DMARC filter checks whether the <recipient, list-id> pair matches an active agreement, for each local recipient. If the message is thus found to be part of an agreed-upon mail flow, the DMARC filter MUST exempt it from being subjected to the DMARC policy published by the message's From: domain. If the receiving domain sends aggregate reports, messages so exempted MUST be counted against the trusted_forwarder PolicyOverrideType.

After setting up DMARC exemptions, the receiving domain sends an acceptance message to the forwarder's base address. From that point on, the forwarder can stop From: munging messages destined for that subscriber. The agreement lasts until it is canceled or removed by the receiving domain. It should be canceled when the corresponding mail flow stops. The receiving domain may send a cancellation message to the base, to unsubscribe the user and terminate the agreement. However, the forwarder has no way to cancel an existing agreement at the receiving domain if the user unsubscribes directly. That is how agreements can become stale. The receiving domain MAY send renewal messages to check whether the mail flow is still active. Stale agreements can then be removed.

Upon receiving a new agreement request with the same coordinates as an existing agreement, the existing agreement SHOULD be considered stale and silently removed before proceeding (unless it is a concurrency bug). Conversely, a cancellation should imply removal, or at least questioning the forwarding itself, be it a mailing list or an alias.

7. Messages to the Base

There are five types of messages that can be sent from the receiving domain to the forwarder's base address. These messages define the status of the forwarding agreement. This status is determined at the sole discretion of the receiving domain. The forwarder can respond affirmatively or negatively. In the event of a bounce or no response, the receiving domain SHOULD resend the message once or twice. After a few days without a response, the agreement can be considered stale and removed.

The message types are as follows:

base-check This message simply checks that the address exists. It can be sent after receiving a web request to confirm that it was received. It is OPTIONAL, as the HTTP return code is sufficient for this purpose. It is not an error to send it at any other time, even if it is useless.

acceptance The receiving domain accepts the agreement. From the moment this message is received onwards, messages can avoid From: munging and, if allowed by the receiving domain, avoid rewriting the bounce address.

rejection The receiving domain, presumably after the user disavows, denies the agreement. Unauthenticated messages may then be rejected or quarantined according to DMARC. The forwarding itself should be questioned. A new agreement SHOULD NOT be requested again unless further interaction with the user occurs.

renewal The receiving domain needs confirmation that this forwarding mechanism is still active. The forwarder must verify that the mechanism corresponding to the agreement is still active before responding. A bounce or no response may result in the agreement being removed.

cancellation The agreement is canceled. For mailing lists, the user should be unsubscribed if not already done. The alias should be removed.

These messages are plain text messages with no attachments and no MIME multipart entities. The agreement-id and the message type are repeated in the Subject: and at the beginning of the body. The rest of the body MAY contain any text.

ABNF:

```
msg-subject       = msg-tag WSP agreement-id ":" [WSP] msg-type
msg-tag           = "[FixForwarding]"
msg-body          = agreement-id-line msg-type-line msg-rest
agreement-id-line = [CRLF] "agreement-id" ":" [WSP] agreement-id CRLF
msg-type-line     = [CRLF] "deal" ":" [WSP] msg-type CRLF
msg-rest          = *OCTET
msg-type          = %s"base-check" / %s"acceptance" / %s"rejection" /
                    %s"renewal" / %s"cancellation"
agreement-id      = "<" id-left "@" id-right ">"
subject           = "Subject" ":" msg-subject CRLF
message           = fields CRLF msg-body
```

Where WSP, CRLF and OCTET are imported from [RFC5234]; subject, id-left, id-right and fields from [RFC5322].

Replies MUST also be text/plain, retaining the Subject: and body of the original message, possibly prefixed as is customary for Internet mail, for example by prefixing the subject with "Re:" and body text lines with "> ". Replies MUST have the In-Reply-To: and References: fields set correctly. The receiving domain can use these to match replies to the original message. For negative replies the first word in the body, and possibly the Subject: MUST be "NO".

ABNF:

```
negative-reply  = fields CRLF %s"NO" CRLF msg-body
```

Both messages and replies MUST be properly authenticated via DKIM and DMARC.

Negative responses are sent when the forwarder is not committed by the agreement. This means the agreement has been previously canceled, has never been archived properly, and is in any case not active. After a negative response, the receiving domain can remove the agreement without further notice.

Responses that are not negative responses are positive acknowledgements of the deal. Responses to cancellation and rejection have the same effect, regardless of whether they are positive or negative, i.e. either party may remove any reference to the agreement as a result. Negative responses to acceptance should

not occur unless the web request was a forgery. Renewal messages, on the other hand, expect a negative response to eventually end the agreement.

8. One ARC Set

In some cases, messages entering an ADministrative Management Domain (ADMD), also known as Organizational Domain, pass through multiple hosts before they are possibly forwarded. Since the message flow within the domain is controlled by the same team, it is useless to add ARC sets during internal transfers.

As described by [RFC7001], the Authentication-Results: header fields are typically set by bastion hosts when they receive external messages. These fields are intended for internal use and are neither signed nor encypted. In order for them to be used and trusted internally, any malicious fields with the same name are scrubbed on ingress by the bastion hosts themselves.

When forwarding a message to external hosts, the ARC sealer collects all trusted Authentication-Results: fields in the message in question and places them into an ARC-Authentication-Results: fields, optionally removing comments.

9. IANA Considerations

Per [RFC8552], please add the following entry to the "Underscored and Globally Scoped DNS Node Names" registry:

RR Type	_NODE NAME	Reference
TXT	_fixforwarding	[This.I-D]

Figure 1

10. Privacy Considerations

Forwarding agreements expose users’ mailing list memberships as well as the final identities of forwarded messages to the administrators of their mailbox provider. This is unavoidable. However, it should be noted that such knowledge can hardly be kept secret from the people who have access to all of their mail. Users must trust their mailbox provider, and this protocol is not the only reason they should do so. However, in situations where the user wishes to hide their forwarding from the receiving domain, they SHOULD NOT request a forwarding agreement. Instead, rewriting the bounce address and

From: header fields, in addition to improving deliverability, can also help hide the true origin of messages.

11. Security Considerations

Forwarders SHOULD check DMARC and enforce author’s domain’s policies whenever possible. In particular, mailing lists, unless they themselves collect posts from other external mailing lists, should reject messages from domains that publish p=reject if authentication fails. Receiving domains cannot reject messages belonging to an accepted agreement, even for DMARC policy reasons reported by Arc-Authentication-Results:, because mailing lists will unsubscribe the user after a few bounces. DMARC Filtering MUST be applied by the forwarder. Failure to apply SHOULD be reported to the forwarder’s abuse address.

12. Acknowledgments

The author would like to thank Murray S. Kucherawy for reviewing the draft and Stephen J. Turnbull for devising the method reported in Appendix A.

13. References

13.1. Normative References

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <https://www.rfc-editor.org/info/rfc2045>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2919] Chandhok, R. and G. Wenger, "List-Id: A Structured Field and Namespace for the Identification of Mailing Lists", RFC 2919, DOI 10.17487/RFC2919, March 2001, <https://www.rfc-editor.org/info/rfc2919>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, DOI 10.17487/RFC5321, October 2008, <https://www.rfc-editor.org/info/rfc5321>.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, DOI 10.17487/RFC5322, October 2008, <https://www.rfc-editor.org/info/rfc5322>.

[RFC5782] Levine, J., "DNS Blacklists and Whitelists", RFC 5782, DOI 10.17487/RFC5782, February 2010, <https://www.rfc-editor.org/info/rfc5782>.

[RFC6376] Crocker, D., Ed., Hansen, T., Ed., and M. Kucherawy, Ed., "DomainKeys Identified Mail (DKIM) Signatures", STD 76, RFC 6376, DOI 10.17487/RFC6376, September 2011, <https://www.rfc-editor.org/info/rfc6376>.

[RFC7489] Kucherawy, M., Ed. and E. Zwicky, Ed., "Domain-based Message Authentication, Reporting, and Conformance (DMARC)", RFC 7489, DOI 10.17487/RFC7489, March 2015, <https://www.rfc-editor.org/info/rfc7489>.

[RFC7578] Masinter, L., "Returning Values from Forms: multipart/form-data", RFC 7578, DOI 10.17487/RFC7578, July 2015, <https://www.rfc-editor.org/info/rfc7578>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8552] Crocker, D., "Scoped Interpretation of DNS Resource Records through "Underscored" Naming of Attribute Leaves", BCP 222, RFC 8552, DOI 10.17487/RFC8552, March 2019, <https://www.rfc-editor.org/info/rfc8552>.

[RFC8617] Andersen, K., Long, B., Ed., Blank, S., Ed., and M. Kucherawy, Ed., "The Authenticated Received Chain (ARC) Protocol", RFC 8617, DOI 10.17487/RFC8617, July 2019, <https://www.rfc-editor.org/info/rfc8617>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

13.2. Informative References

[RFC5598] Crocker, D., "Internet Mail Architecture", RFC 5598, DOI 10.17487/RFC5598, July 2009, <https://www.rfc-editor.org/info/rfc5598>.

[RFC6377] Kucherawy, M., "DomainKeys Identified Mail (DKIM) and Mailing Lists", BCP 167, RFC 6377, DOI 10.17487/RFC6377, September 2011, <https://www.rfc-editor.org/info/rfc6377>.

[RFC7001] Kucherawy, M., "Message Header Field for Indicating Message Authentication Status", RFC 7001, DOI 10.17487/RFC7001, September 2013, <https://www.rfc-editor.org/info/rfc7001>.

[RFC7208] Kitterman, S., "Sender Policy Framework (SPF) for Authorizing Use of Domains in Email, Version 1", RFC 7208, DOI 10.17487/RFC7208, April 2014, <https://www.rfc-editor.org/info/rfc7208>.

[RFC7960] Martin, F., Ed., Lear, E., Ed., Draegen, T., Ed., Zwicky, E., Ed., and K. Andersen, Ed., "Interoperability Issues between Domain-based Message Authentication, Reporting, and Conformance (DMARC) and Indirect Email Flows", RFC 7960, DOI 10.17487/RFC7960, September 2016, <https://www.rfc-editor.org/info/rfc7960>.

[SRS] "SRS", <https://en.wikipedia.org/wiki/Sender_Rewriting_Scheme>.

Appendix A. No-Munging Method

This protocol assumes that From: munging can be enabled or disabled on a per-user basis. Many MLMs instead provide an option to enable or disable it on a per-list basis. Here is a way to overcome this limitation.

Let’s say you have an umbrella list with two sibling sub-lists, one configured with From: munging, the other one without. Both sub-lists refuse all posts, and advertise the umbrella list as the post address.

The umbrella list accepts posts according to site and list policy. It has the sibling sub-lists as its only subscribers, and won’t accept more.

The sibling sub-list that does From: munging accepts subscribers under the site and list policy. When forwarding agreements are accepted, the relevant subscribers are moved to the other sub-list.

Appendix B. Tiny Provider Example

A personal or family domain that has its own mail server can participate in fixforwarding agreements by setting up a web form that mails submissions to the admin email address, which is also the base address. Then it publishes a _fixforwarding DNS record that sets the form's URL as post=.

When someone submits a web request, the admin can easily check if it is good, since he or she knows each user personally. Evil submission don't really need an explicit rejection. For good ones, sending a acceptance response to the base address of the request can be done via any mail client. The same applies to renewal or cancellation, upon user request.

Before sending positive responses, the admin MUST add the forwarder's data, domain and list-id, to the DMARC filter whitelist. Having a filter that accepts such a configuration is the only special software requisite. Since there are no malicious users in a home environment, this domain can decide to trust a <forwarder, list-id> pair for all users after one of them has confirmed. It is also possible to trust a forwarder domain regardless of the list-id. This type of decision simplifies the DMARC filter settings, but not agreement management.

Appendix C. Tiny Forwarder Example

A user of the above family mail server may want some or all of her messages to be forwarded to an external address. When setting up a dot-forward file for the user, the admin can check whether the target domain has a _fixforwarding record. If so, the related web form can be filled out manually, specifying the admin's own address as the base and abuse fields. As long as renewal requests arrive every few months, they are a useful reminder; otherwise an autoresponder will be in order.

The admin address can also be substituted for the bounce address of forwarded messages. This way, the user can be notified in case the remote mailbox gets full. ARC signatures are added to all forwarded messages anyway, after adding the designated List-ID: that was specified in the web form. There is not much that can be done for bounces due to DMARC policy. However, once the remote domain has accepted the forwarding request, they should not happen again.

Appendix D. Large Provider Example

Request processing involves interaction between the receiving domain and its user who initiated the forwarding request with the forwarder. The key is to present a clear prompt to the user so that a consistent confirmation or denial is obtained. Asking the user for confirmation can also be an opportunity to manage mailbox details such as which folder or label the user wants to associate with the new mail flow. Mailbox providers can also implement a web application that lists the status of all forwarding agreements in place; this would be somewhat more reliable than monthly subscription reminders, which some call a time-distributed database.

The web form should be set to reject submissions from blacklisted IP addresses or referring to known bad domains. However, it cannot use captchas. A domain that suffers an overwhelming number of bogus requests may resort to authorizations. The request for authorization may be subject to captchas or even disclosure of the requester's identity. Once granted, authorization should remain valid for all requests with the same abuse address.

Accepted requests can be saved on MX servers, as <list-id> files (containing the signing domain) in each user's home folder, or as <user, list-id> pairs in a database, or in any other way convenient for the DMARC filter.

Sending renewal messages to the base can be done every few months in order to clean up the storage of unused entries. Cancellation may never be used, since it is more practical to use the List-Unsubscribe: header field of a sample message to deactivate the flow. However, if the provider implements a web application that lists the status of all agreements in place, it becomes more convenient to send a cancellation message to the base to delete one of these entries.

Appendix E. Mailing List Example

During the sunrise period, a MLM can monitor its subscribers' domains to see if any of them have adopted this protocol.

A mailing list may want to check whether a _fixforwarding DNS record exists for a subscriber's domain before sending a COI. If the domain is known to handle fixforwarding requests correctly, an acceptance to the base is semantically equivalent. Otherwise, the web request can be issued after the COI has been confirmed, and the COI itself can mention that further confirmation from the provider may follow. After an acceptance mail has been received at the base, From: munging for that subscriber can be suspended.

Implementing an auto-responder for the base shouldn’t be difficult. The base address can be an alias for each subscriber, so that they automatically bounce after cancellation.

Appendix F. Web Form Example

Mail operators won’t manually fill web forms; see next appendix. However occasional forwarders of small mail sites or those trying it for the first time may want to do so. Here is an example PHP code to receive a request and post it to a temporary file, which someone monitoring that directory will then act on.

```
<?php

function common_end()
{
    print <<<END_OF_FOOTER
</body>
</html>
END_OF_FOOTER;
}

    $mystyle = ’
<style>
    form {
      display: block;
      /* Form outline */
      width: 100%
    }

    label {
      /* Uniform size & alignment */
      display: block;
      min-width: 90px;
      text-align: left;
    }

    input,
    textarea {
      /* Make sure that all text fields have the same font settings
         By default, text areas have a monospace font */
      font: 1em sans-serif;
      /* Uniform text field size */
      width: 100%;
      box-sizing: border-box;
      /* Match form field borders */
      border: 1px solid #999999;
      margin-bottom: 1em;
```

```
       }

       input:focus,
       textarea:focus {
         /* Set the outline width and style */
         outline-style: solid;
         /* To give a little highlight on active elements */
         outline-color: black;
       }

       textarea {
         /* Align multiline text fields with their labels */
         vertical-align: top;
         /* Provide space to type some text */
         height: 5em;
       }

       .button {
         /* Align buttons with the text fields */
         padding-left: 90px; /* same size as the label elements */
       }

       button {
         /* This extra margin represent roughly the same space as the
            space between the labels and their text fields */
         margin-left: 0.5em;
       }
   </style>’;

       $PHP_SELF = $_SERVER[’PHP_SELF’];

       $values = 0;
       $valid = 0;

       /*
       * 9 fields:  verify each value individually
       */
       $abuse_check = 0;
       $abuse = @$_POST["abuse"];
       if (strlen($abuse))
       {
           ++$values;
           if (filter_var($abuse, FILTER_VALIDATE_EMAIL))
           {
               $abuse_check = 1;
               ++$valid;
           }
           else
```

```
            $abuse_check = -1;
        }

        $agreement_id_check = 0;
        $agreement_id = @$_POST["agreement-id"];
        if (strlen($agreement_id))
        {
            ++$values;
            if (filter_var($agreement_id, FILTER_VALIDATE_EMAIL))
            {
                $agreement_id_check = 1;
                ++$valid;
            }
            else
                $agreement_id_check = -1;
        }

        $base_check = 0;
        $base = @$_POST["base"];
        if (strlen($base))
        {
            ++$values;
            if (filter_var($base, FILTER_VALIDATE_EMAIL))
            {
                $base_check = 1;
                ++$valid;
            }
            else
                $base_check = -1;
        }

        $collector_check = 0;
        $collector = @$_POST["collector"];
        if (strlen($collector))
        {
            ++$values;
            if (filter_var($collector, FILTER_VALIDATE_EMAIL))
            {
                $collector_check = 1;
                ++$valid;
            }
            else
                $collector_check = -1;
        }

        $domain_check = 0;
        $domain = @$_POST["domain"];
        if (strlen($domain))
```

```
    {
        ++$values;
        if (filter_var($domain, FILTER_VALIDATE_DOMAIN))
        {
            $domain_check = 1;
            ++$valid;
        }
        else
            $domain_check = -1;
    }

    $emitter_check = 0;
    $emitter = @$_POST["emitter"];
    if (strlen($emitter))
    {
        ++$values;
        if (filter_var($emitter, FILTER_VALIDATE_EMAIL))
        {
            /*
            * Check that $emitter is a local user.
            * Basically, the domain-part of the address must be one
            * of the hosted domain, and an SMTP session attempting to
            * send an email must pass the RCPT command.  Since this
            * depends on the mail server software, it is delegated to
            * an external script.
            */
            $output = null;
            $retval = null;
            exec('verify_emitter.sh '.
                escapeshellarg($emitter).' 2>&1',
                $output, $retval);
            if ($retval === 0 && $output[0] === 'ok')
            {
                $emitter_check = 1;
                ++$valid;
            }
            else
                $emitter_check = -1;
        }
        else
            $emitter_check = -1;
    }

    $list_id_check = 0;
    $list_id = @$_POST["list-id"];
    if (strlen($list_id))
    {
        ++$values;
```

```
            if (filter_var($list_id, FILTER_VALIDATE_DOMAIN))
            {
                $list_id_check = 1;
                ++$valid;
            }
            else
                $list_id_check = -1;
        }

        $timeout_check = 0;
        $timeout = @$_POST["timeout"];
        if (strlen($timeout))
        {
            ++$values;
            if (filter_var($timeout, FILTER_VALIDATE_INT,
                array("options" => array("min_range" => 86400))))
            {
                $timeout_check = 1;
                ++$valid;
            }
            else
                $timeout_check = -1;
        }

        $text_check = 0;
        $text = @$_POST["text"];
        if (strlen($text))
        {
            ++$values;
            /*
            * Check the text to send to the user contains no links
            * and no tags.  Split the regex for editorial reasons.
            */
            $http_preg_host = '@(https?://([-\w\.]+[-\w])+(:\d+)?';
            $http_preg_param = '(/([\w/_\.#-]*(\?\S+)?[^\.\s])?)?)@';
            $http_preg = $http_preg_host . $http_preg_param;
            $filtered = preg_replace($http_preg, "", strip_tags($text));
            if (strcmp($filtered, $text) === 0)
            {
                $text_check = 1;
                ++$valid;
            }
            else
                $text_check = -1;
        }

        /*
        * Check that List-ID matches domain
```

```
        */
        if ($domain_check > 0 && $list_id_check > 0)
        {
            if (!str_ends_with($list_id, '.'. $domain))
            {
                $domain_check = -1;
                $list_id_check = -1;
                $valid -= 2;
            }
        }

        print <<<END_OF_TEXT
    <!DOCTYPE HTML>
    <head>
    <meta http-equiv="Content-type" content="text/html; charset=UTF-8">
    <title>Forwarding Authorization Request</title>
    <link href="/favicon.ico" rel="shortcut icon">$mystyle
    </head>
    <body>
    <h1 style="text-align: center;">Forwarding Authorization Request</h1>

    END_OF_TEXT;

        if ($values == 9 && $valid == 9)
        {
            ignore_user_abort(TRUE);
            $save = json_encode(array(
                "abuse" => $abuse,
                "agreenment_id" => $agreement_id,
                "base" => $base,
                "collector" => $collector,
                "domain" => $domain,
                "emitter" => $emitter,
                "list_id" => $list_id,
                "timeout" => $timeout,
                "text" => $text), JSON_PRETTY_PRINT);
            if (($fname = tempnam('/var/tmp/', 'fixfor')) === false ||
                file_put_contents($fname, $save) === false)
            {
                error_log("$PHP_SELF: ". var_dump($save), 0);
                http_response_code(500);
                print "
                    <p>
                    Internal error while saving data.
                    Please try again later and/ or write to postmaster
                ";
            }
            else
```

```
            {
                http_response_code(202);
                print "
                    <p>
                    Request received.
                    $base will be contacted soon.
                ";
            }
            print "
                </div>";
            common_end();
            flush();
            exit();
        }

        // Print form with any existing value
        print '
            <form action="'. $PHP_SELF .'" method="POST">
            <fieldset><legend>Fixforwarding Form</legend>
            ';
        printf('
            <label for="domain">%sDomain requesting agreement</label>
            <input type="text" id="domain" name="domain"%s>',
            $domain_check < 0? 'INVALID: ': '',
            strlen($domain)? ' value="'. $domain .'"': '');
        printf('
            <label for="abuse">
            %sAbuse team email address of requesting domain</label>
            <input type="email" id="abuse" name="abuse"%s>',
            $abuse_check < 0? 'INVALID: ': '',
            strlen($abuse)? ' value="'. $abuse .'"': '');
        printf('
            <label for="agreement-id">%sAgreement ID</label>
            <input type="email" id="agreement-id"
                name="agreement-id"%s>',
            $agreement_id_check < 0? 'INVALID: ': '',
            strlen($agreement_id)? ' value="'. $agreement_id .'"': '');
        printf('
            <label for="list-id">
            %sList-ID header for all forwarded messages</label>
            <input type="text" id="list-id" name="list-id"%s>',
            $list_id_check < 0? 'INVALID: ': '',
            strlen($list_id)? ' value="'. $list_id .'"': '');
        printf('
            <label for="base">
            %sBase email address for managing this agreement</label>
            <input type="email" id="base" name="base"%s>',
            $base_check < 0? 'INVALID: ': '',
```

```
            strlen($base)? ' value="'. $base .'"': '');
        printf('
            <label for="collector">
            %sCollector: list post address or alias</label>
            <input type="email" id="collector" name="collector"%s>',
            $collector_check < 0? 'INVALID: ': '',
            strlen($collector)? ' value="'. $collector .'"': '');
        printf('
            <label for="emitter">
            %sEmitter: final recipient email address</label>
            <input type="email" id="emitter" name="emitter"%s>',
            $emitter_check < 0? 'INVALID: ': '',
            strlen($emitter)? ' value="'. $emitter .'"': '');
        printf('
            <label for="text">
            %sText to send to the user to confirm authorization</label>
            <textarea id="text" name="text" rows="6" cols="40"
                maxlength="4096">%s</textarea>',
            $text_check < 0? 'INVALID: no tags and no links': '',
            strlen($text)? $text: '');
        printf('
            <label for="timeout">
            %sTime to wait for user response (seconds)</label>
            <input type="number" id="timeout"
                name="timeout" min="86400"%s>',
            $timeout_check < 0? 'INVALID: ': '',
            strlen($timeout)? ' value="'. $timeout .'"': '');
        print '
            <br>
            <input type="submit">
            </fieldset></form></div>';

        common_end();
        flush();
        exit();
    ?>
```

Appendix G. Automatic Submission Example

A mail operator can easily determine if the fixforwarding protocol can be used for a given email address by examining the underscored _fixforwarding DNS record. This example only shows sending the request. It doesn't show how to get the new subscriber's address, how to lookup and examine the DNS record, nor how to generate an ID. Most fields are fixed for this domain. A "file.txt" file contains the text to be presented to the user. This example assumes that a new mail alias, equal to the ID, is created for each new request; this may ease some operations.

```
   # address of the new subscriber
   TARGET=user@example.org

   # check _fixforwarding.example.org TXT record
   # and  extract the post address
   POST=https://www.example.org/.well-known/fixforwarding.php

   # generate a new ID
   NEWID=ffid_dcac377fb673b0bfd9e641894d06bd3a51c1c732

   rtc=$(curl --no-progress-meter              \
       -d domain=example.com                   \
       -d abuse=abuse@example.com              \
       -d agreement-id=$NEWID@example.com      \
       -d list-id=$NEWID.example.com           \
       -d base=$NEWID@example.com              \
       -d collector=mylist@example.com         \
       -d emitter=$TARGET                      \
       --data-urlencode text@file.txt          \
       -d timeout=604800                       \
       --write-out '%{http_code}' -o /dev/null \
       $POST)
   if [ -n "$rtc" -a $rtc -eq 202 ]; then
       echo "Request for $TARGET succeeded"
   fi
```

Author's Address

Alessandro Vesely
via L. Anelli 13
20122 Milano MI
Italy
Email: vesely@tana.it