APPSAWG
Internet-Draft
Intended status: Informational
Expires: April 12, 2013

M. Kucherawy
G. Shapiro

October 9, 2012

Advice for Safe Handling of Malformed Messages
draft-ietf-appsawg-malformed-mail-03

Abstract

The email ecosystem has long had a very permissive set of common processing rules in place, despite increasingly rigid standards governing its components, ostensibly to improve the user experience. The handling of these come at some cost, and various components are faced with decisions about whether or not to permit non-conforming messages to continue toward their destinations unaltered, adjust them to conform (possibly at the cost of losing some of the original message), or outright rejecting them.

This document includes a collection of the best advice available regarding a variety of common malformed mail situations, to be used as implementation guidance. It must be emphasized, however, that the intent of this document is not to standardize malformations or otherwise encourage their proliferation. The messages are manifestly malformed, and the code and culture that generates them needs to be fixed. Therefore, these messages should be rejected outright if at all possible. Nevertheless, many malformed messages from otherwise legitimate senders are in circulation and will be for some time, and, unfortunately, commercial reality shows that we cannot always simply reject or discard them. Accordingly, this document presents alternatives for dealing with them in ways that seem to do the least additional harm until the infrastructure is tightened up to match the standards.

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 http://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 April 12, 2013.

Table of Contents

1. Introduction

1.1. The Purpose Of This Work

The history of email standards, going back to [RFC822] and beyond, contains a fairly rigid evolution of specifications. But implementations within that culture have also long had an undercurrent known formally as the robustness principle, but also known informally as Postel's Law: "Be conservative in what you do, be liberal in what you accept from others."

In general, this served the email ecosystem well by allowing a few errors in implementations without obstructing participation in the game. The proverbial bar was set low. However, as we have evolved into the current era, some of these lenient stances have begun to expose opportunities that can be exploited by malefactors. Various email-based applications rely on strong application of these standards for simple security checks, while the very basic building blocks of that infrastructure, intending to be robust, fail utterly to assert those standards.

This document presents some areas in which the more lenient stances can provide vectors for attack, and then presents the collected wisdom of numerous applications in and around the email ecosystem for dealing with them to mitigate their impact.

1.2. Not The Purpose Of This Work

It is important to understand that this work is not an effort to endorse or standardize certain common malformations. The code and culture that introduces such messages into the mail stream needs to be repaired, as the security penalty now being paid for this lax processing arguably outweighs the reduction in support costs to end users who are not expected to understand the standards. However, the reality is that this will not be fixed quickly.

Given this, it is beneficial to provide implementers with guidance about the safest or most effective way to handle malformed messages when they arrive, taking into consideration the tradeoffs of the choices available especially with respect to how various actors in the email ecosystem respond to such messages in terms of handling, parsing, or rendering to end users.

1.3. General Considerations

Many deviations from message format standards are considered by some receivers to be strong indications that the message is undesirable, i.e., is spam or contains malware. Such receivers quickly decide

that the best handling choice is simply to reject or discard the message. This means malformations caused by innocent misunderstandings or ignorance of proper syntax can cause messages with no ill intent also to fail to be delivered.

Senders that want to ensure message delivery are best advised to adhere strictly to the relevant standards (including, but not limited to, [MAIL], [MIME], and [DKIM]), as well as observe other industry best practices such as may be published from time to time either by the IETF or independently.

2. Document Conventions

2.1. Key Words

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [KEYWORDS]. However, they only have that meaning in this document when they are presented entirely in upper case.

2.2. Examples

Examples of message content include a number within braces at the end of each line. These are line numbers for use in subsequent discussion, and are not actually part of the message content presented in the example.

Blank lines are not numbered in the examples.

3. Background

The reader would benefit from reading [EMAIL-ARCH] for some general background about the overall email architecture. Of particular interest is the Internet Message Format, detailed in [MAIL]. Throughout this document, the use of the term "messsage" should be assumed to mean a block of text conforming to the Internet Message Format.

4. Internal Representations

Any agent handling a message could have one or two (or more) distinct representations of a message it is handling. One is an internal representation, such as a block of storage used for the header and a block for the body. These may be sorted, encoded, decoded, etc., as per the needs of that particular module. The other is the representation that is output to the next agent in the handling chain. This might be identical to the version that is input to the

module, or it might have some changes such as added or reordered header fields, body modifications to remove malicious content, etc.

In some cases, advice is provided only for internal representations. However, there is often occasion to mandate changes to the output as well.

5. Invariate Content

Experience has shown that it is beneficial to ensure that, from the first analysis agent at ingress into the destination Administrative Management Domain (ADMD; see [EMAIL-ARCH]) to the agent that actually affects delivery to the end user, the message each agent sees is identical. Absent this, it can be impossible for different agents in the chain to make assertions about the content that correlate.

For example, suppose a handling agent records that a message had some specific set of properties at ingress to the ADMD, then permitted it to continue inbound. Some other agent alters the content for some reason. The user, on viewing the delivered content, reports the message as abusive. If the report is based on the set of properties recorded at ingress, then the complaint effectively references a message different from what the user saw, which could render the complaint inactionable. Similarly, a message with properties that a filtering agent might use to reject an abusive message could be allowed to reach the user if an intermediate agent altered the message in a manner that alters one of those properties, thwarting detection of the abuse.

Therefore, agents comprising an inbound message processing environment SHOULD ensure that each agent sees the same content, and the message reaches the end user unmodified. An exception to this is content that is identitfied as certainly harmful, such as some kind of malicious executable software included in the message.

6. Mail Submission Agents

Within the email context, the single most influential component that can reduce the presence of malformed items in the email system is the Mail Submission Agent (MSA). This is the component that is essentially the interface between end users that create content and the mail stream.

The lax processing described earlier in the document creates a high support and security cost overall. Thus, MSAs MUST evolve to become more strict about enforcement of all relevant email standards, especially [MAIL] and the [MIME] family of documents.

Relay Mail Transport Agents (MTAs) SHOULD also be more strict; although preventing the dissemination of malformed messages is desirable, the rejection of such mail already in transit also has a support cost, namely the creation of a [DSN] that many end users might not understand.

7. Line Terminaton

The only valid line separation sequence in messaging is ASCII 0x0D ("carriage return", or CR) followed by ASCII 0x0A ("line feed", or LF), commonly referred to as CRLF. Common UNIX user tools, however, typically only use LF for line termination. This means the protocol has to convert LF to CRLF before transporting a message.

Naive implementations can cause messages to be transmitted with a mix of line terminations, such as LF everywhere except CRLF only at the end of the message. According to [SMTP], this means the entire message actually exists on a single line.

A "naked" CR or LF in a message has no reasonable justification, and furthermore [MIME] presents mechanisms for encoding content that actually does need to contain such an unusual character sequence.

Thus, handling agents MUST treat naked CRs and LFs as CRLFs when interpreting the message.

8. Header Anomalies

This section covers common syntactical and semantic anomalies found in headers of messages, and presents preferred mitigations.

8.1. Converting Obsolete and Invalid Syntaxes

There are numerous cases of obsolete header syntaxes that can be applied to confound agents with variable processing. This section presents some examples of these. Messages including them SHOULD be rejected; where this is not possible, RECOMMENDED internal interpretations are provided.

8.1.1. Host-Address Syntax

The following obsolete syntax:

```
To: <@example.net:fran@example.com>
```

should be interpreted as:

```
To: <fran@example.com>
```

8.1.2. Excessive Angle Brackets

The following over-use of angle brackets, e.g.:

```
To: <<<user2@example.org>>>
```

should be interpreted as:

```
To: <user2@example.org>
```

8.1.3. Unbalanced Angle Brackets

The following use of unbalanced angle brackets:

```
To: <another@example.net
To: second@example.org>
```

should be interpreted as:

```
To: <another@example.net>
To: second@example.org
```

8.1.4. Unbalanced Parentheses

The following use of unbalanced parentheses:

```
To: (Testing <fran@example.com>
To: Testing) <sam@example.com>
```

should be interpreted as:

```
To: (Testing) <fran@example.com>
To: "Testing)" <sam@example.com>
```

8.1.5. Unbalanced Quotes

The following use of unbalanced quotation marks:

```
To: "Joe <joe@example.com>
```

should be interpreted as:

```
To: "Joe <joe@example.com>"@example.net
```

where "example.net" is the domain name or host name of the handling agent making the interpretation.

8.2. Non-Header Lines

It has been observed that some messages contain a line of text in the header that is not a valid message header field of any kind. For example:

```
From: user@example.com {1}
To: userpal@example.net {2}
Subject: This is your reminder {3}
about the football game tonight {4}
Date: Wed, 20 Oct 2010 20:53:35 -0400 {5}

Don't forget to meet us for the tailgate party! {7}
```

The cause of this is typically a bug in a message generator of some kind. Line {4} was intended to be a continuation of line {3}; it should have been indented by whitespace as set out in Section 2.2.3 of [MAIL].

This anomaly has varying impacts on processing software, depending on the implementation:

1. some agents choose to separate the header of the message from the body only at the first empty line (i.e. a CRLF immediately followed by another CRLF);

2. some agents assume this anomaly should be interpreted to mean the body starts at line {4}, as the end of the header is assumed by encountering something that is not a valid header field or folded portion thereof;

3. some agents assume this should be interpreted as an intended header folding as described above and thus simply append a single space character (ASCII 0x20) and the content of line {4} to that of line {3};

4. some agents reject this outright as line {4} is neither a valid header field nor a folded continuation of a header field prior to an empty line.

This can be exploited if it is known that one message handling agent will take one action while the next agent in the handling chain will take another. Consider, for example, a message filter that searches message headers for properties indicative of abusive of malicious content that is attached to a Mail Transfer Agent (MTA) implementing option 2 above. An attacker could craft a message that includes this malformation at a position above the property of interest, knowing the MTA will not consider that content part of the header, and thus

the MTA will not feed it to the filter, thus avoiding detection. Meanwhile, the Mail User Agent (MUA) which presents the content to an end user, implements option 1 or 3, which has some undesirable effect.

It should be noted that a few implementations choose option 4 above since any reputable message generation program will get header folding right, and thus anything so blatant as this malformation is likely an error caused by a malefactor.

The preferred implementation if option 4 above is not employed is to apply the following heuristic when this malformation is detected:

1. Search forward for an empty line. If one is found, then apply option 3 above to the anomalous line, and continue.

2. Search forward for another line that appears to be a new header field, i.e., a name followed by a colon. If one is found, then apply option 3 above to the anomalous line, and continue.

8.3. Unusual Spacing

The following message is valid per [MAIL]:

```
From: user@example.com {1}
To: userpal@example.net {2}
Subject: This is your reminder {3}
 {4}
 about the football game tonight {5}
Date: Wed, 20 Oct 2010 20:53:35 -0400 {6}

Don't forget to meet us for the tailgate party! {8}
```

Line {4} contains a single whitespace. The intended result is that lines {3}, {4}, and {5} comprise a single continued header field. However, some agents are aggressive at stripping trailing whitespace, which will cause line {4} to be treated as an empty line, and thus the separator line between header and body. This can affect header-specific processing algorithms as described in the previous section.

Ideally, this case simply ought not to be generated.

Message handling agents receiving a message bearing this anomaly MUST behave as if line {4} was not present on the message, and SHOULD emit a version in which line {4} has been removed.

8.4. Header Malformations

There are various malformations that exist. A common one is insertion of whitespace at unusual locations, such as:

```
From: user@example.com {1}
To: userpal@example.net {2}
Subject: This is your reminder {3}
MIME-Version : 1.0 {4}
Content-Type: text/plain {5}
Date: Wed, 20 Oct 2010 20:53:35 -0400 {6}

Don't forget to meet us for the tailgate party! {8}
```

Note the addition of whitespace in line {4} after the header field name but before the colon that separates the name from the value.

The acceptance grammar of [MAIL] permits that extra whitespace, so it cannot be considered invalid. However, a consensus of implementations prefers to remove that whitespace. There is no perceived change to the semantics of the header field being altered as the whitespace is itself semantically meaningless. Thus, a module compliant with this memo MUST remove all whitespace after the field name but before the colon, and MUST emit that version of that field on output.

8.5. Header Field Counts

Section 3.6 of [MAIL] prescribes specific header field counts for a valid message. Few agents actually enforce these in the sense that a message whose header contents exceed one or more limits set there are generally allowed to pass; they may add any required fields that are missing, however.

Also, few agents that use messages as input, including Mail User Agents (MUAs) that actually display messages to users, verify that the input is valid before proceeding. Two popular open source filtering programs and two popular Mailing List Management (MLM) packages examined at the time this document was written select either the first or last instance of a particular field name, such as From, to decide who sent a message. Absent enforcement of [MAIL], an attacker can craft a message with multiple fields if that attacker knows the filter will make a decision based on one but the user will be shown the other.

This situation is exacerbated when a claim of message validity is inferred by something like a valid [DKIM] signature. Such a signature might cover one instance of a constrained field but not

another, and a naive consumer of DKIM's output, not realizing which one was covered by a valid signature, could presume the wrong one was the "good" one. An MUA, for example could show the first of two From fields as "good" or "safe" while the DKIM signature actually only verified the second.

Thus, an agent compliant with this specification MUST enact one of the following:

1. reject outright or refuse to process further any input message that does not conform to Section 3.6 of [MAIL];

2. remove or, in the case of an MUA, refuse to render any instances of a header field whose presence exceeds a limit prescribed in Section 3.6 of [MAIL] when generating its output;

3. alter the name of any header field whose presence exceeds a limit prescribed in Section 3.6 of [MAIL] when generating its output so that later agents can produce a consistent result. Any alteration likely to cause the field to be ignored by downstream agents is acceptable. A common approach is to prefix the field names with a string such as "BAD-".

8.6. Missing Header Fields

Similar to the previous section, there are messages seen in the wild that lack certain required header fields. For example, [MAIL] requires that a From and Date field be present in all messages.

When presented with a message lacking these fields, the MTA might perform one of the following:

1. Make no changes

2. Add an instance of the missing field(s) using synthesized content

Option 2 is RECOMMENDED for handling this case. Handling agents SHOULD add these for internal hanlding if they are missing, but MUST NOT add them to the external representation. The reason for this requirement is that there are some filter modules that would consider the absence of such fields to be a condition warranting special treatment (e.g., rejection), and thus the effectiveness of such modules would be stymied by an upstream filter adding them.

The synthesized fields SHOULD contain a best guess as to what should have been there; for From, the SMTP MAIL command's address can be used (if not null) or a placeholder address followed by an address literal (e.g., unknown@[192.0.2.1]); for Date, a date extracted from

a Received field is a reasonable choice.

One other important case to consider is a missing Message-Id field. An MTA that encounters a message missing this field SHOULD synthesize a valid one using techniques described above and add it to the external rpresentation, since many deployed tools use the content of that field as a common unique message reference, so its absence inhibits correlation of message processing. One possible synthesis would be based on based on an encoding of the current date/time and an internal MTA ID (e.g., queue ID) followed by @ and the fully qualified hostname of the machine synthesizing the header value. For example:

```
tm = gmtime(&now);
(void) snprintf(buf, sizeof(buf), "%04d%02d%02d%02d%02d.%s@%s",
                tm->tm_year + 1900, tm->tm_mon + 1, tm->tm_mday,
                tm->tm_hour, tm->tm_min, queueID, fqhn);
```

8.7. Eight-Bit Data

Standards-compliant mail messages do not contain any non-ASCII data without indicating that such content is present by means of published [SMTP] extensions. Absent that, [MIME] encodings are typically used to convert non-ASCII data to ASCII in a way that can be reversed by other handling agents or end users.

Non-ASCII data otherwise found in messages can confound code that is used to analyze content. For example, a null (ASCII 0x00) byte inside a message can cause typical string processing functions to mis-identify the end of a string, which can be exploited to hide malicious content from analysis processes.

Handling agents MUST reject messages containing null bytes that are not encoded in some standard way, and SHOULD reject other non-ASCII bytes that are similarly not encoded. If rejection is not done, an ASCII-compatible encoding such as those defined in [MIME] SHOULD be used.

9. MIME Anomalies

[MIME], et seq, define a mechanism of message extensions for providing text in character sets other than ASCII, non-text attachments to messages, multi-part message bodies, and similar facilities.

Some anomalies with MIME-compliant generation are also common. This section discusses some of those and presents preferred mitigations.

9.1. Header Field Names

[MAIL] permits header field names to begin with "--". This means that a header field name can look like a [MIME] multipart boundary. For example:

```
  --foo:bar
```

This is a legal header field, whose name is "--foo" and whose value is "bar". Thus, consider this header:

```
From: user@example.com {1}
To: userpal@example.net {2}
Subject: This is your reminder {3}
Date: Wed, 20 Oct 2010 20:53:35 -0400 {4}
MIME-Version: 1.0 {5}
Content-Type: multipart/mixed; boundary="foo:bar" {6}
--foo:bar {7}
Malicious-Content: muahaha {8}
```

One implementation could observe that line {7} announces the beginning of the first MIME part while another considers it a part of the message's header.

If rejection of such messages cannot be done, agents MUST treat line {7} as part of the message's header block and not a MIME boundary.

9.2. Missing MIME-Version Field

Any message that uses [MIME] constructs is required to have a MIME-Version header field. Without them, the Content-Type and associated fields have no semantic meaning.

It is often observed that a message has complete MIME structure, yet lacks this header field.

As described at the end of Section 8.2, this is not expected from a reputable content generator and is often an indication of mass-produced spam or other undesirable messages.

Therefore, an agent compliant with this specification MUST internally enact one or more of the following in the absence of a MIME-Version header field:

1. Ignore all other MIME-specific fields, even if they are syntactically valid, thus treating the entire message as a single-part message of type text/plain;

2. Remove all other MIME-specific fields, even if they are syntactically valid, both internally and when emitting the output version of the message;

10. Body Anomalies

10.1. Oversized Lines

A message containing a line of content that exceeds 998 characters plus the line terminator (1000 total) violates Section 2.1.1 of [MAIL]. Some handling agents may not look at content in a single line past the first 998 bytes, providing bad actors an opportunity to hide malicious content.

There is no specified way to handle such messages, other than to observe that they are non-compliant and reject them, or rewrite the oversized line such that the message is compliant.

Handling agents MUST take one of the following actions:

1. Break such lines into multiple lines at a position that does not change the semantics of the text being thus altered. For example, breaking an oversized line such that a [URI] then spans two lines could inhibit the proper identification of that URI.

2. Rewrite the MIME part (or the entire message if not MIME) that contains the excessively long line using a content encoding that breaks the line in the transmission but would still result in the line being intact on decoding for presentation to the user. Both of the encodings declared in [MIME] can accomplish this.

11. Security Considerations

The discussions of the anomalies above and their prescribed solutions are themselves security considerations. The practises enumerated in this memo are generally perceived as attempts to resolve security considerations that already exist rather than introducing new ones.

12. IANA Considerations

This memo contains no actions for IANA.

[RFC Editor: Please remove this section prior to publication.]

13. References

13.1. Normative References

[KEYWORDS] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[MAIL] Resnick, P., "Internet Message Format", RFC 5322, October 2008.

13.2. Informative References

[DKIM] Allman, E., Callas, J., Delany, M., Libbey, M., Fenton, J., and M. Thomas, "DomainKeys Identified Mail (DKIM) Signatures", RFC 4871, May 2007.

[DSN] Moore, K. and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 3464, January 2003.

[EMAIL-ARCH] Crocker, D., "Internet Mail Architecture", RFC 5598, July 2009.

[MIME] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC822] Crocker, D., "Standard for the Format of Internet Text Messages", RFC 822, August 1982.

[SMTP] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[URI] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", RFC 3986, January 2005.

Appendix A. Acknowledgements

The author wishes to acknowledge the following for their review and constructive criticism of this proposal: Tony Hansen, and Franck Martin

Authors' Addresses

Murray S. Kucherawy

EMail: superuser@gmail.com

Gregory N. Shapiro

EMail: gshapiro@sendmail.com