APPSAWG
Internet-Draft
Intended status: Informational
Expires: March 21, 2014

M. Kucherawy
G. Shapiro
N. Freed

September 17, 2013

Advice for Safe Handling of Malformed Messages
draft-ietf-appsawg-malformed-mail-08

Abstract

Although Internet mail formats have been precisely defined since the 1970s, authoring and handling software often show only mild conformance to the specifications. The distributed and non-interactive nature of email has often prompted adjustments to receiving software, to handle these variations, rather than trying to gain better conformance by senders, since the receiving operator is primarily driven by complaining recipient users and has no authority over the sending side of the system. Processing with such flexibility comes at some cost, since mail software is 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.

A core requirement for interoperability is that both sides of an exchange work from the same details and semantics. By having receivers be flexible, beyond the specifications, there can be -- and often has been -- a good chance that a message will not be fully interoperable. Worse, a well-established pattern of tolerance for variations can sometimes be used as an attack vector.

This document includes a collection of the best advice available regarding a variety of common malformed mail situations, to be used as implementation guidance. These malformations are typically based around loose interpretations or implementations of specifications such as Internet Message Format [MAIL] and Multipurpose Internet Mail Extensions [MIME].

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

Copyright Notice

Table of Contents

1. Introduction

1.1. The Purpose Of This Work

The history of email standards, going back to [RFC733] 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."

Jon Postel's directive is often misinterpreted to mean that any deviance from a specification is acceptable. Rather, it was intended only to account for legitimate variations in interpretation within specifications, as well as basic transit errors, like bit errors. Taken to its unintended extreme, excessive tolerance would imply that there are no limits to the liberties that a sender might take, while presuming a burden on a receiver to guess "correctly" at the meaning of any such variation. These matters are further compounded by flawed receiver software -- the end users' mail readers -- which are also sometimes flawed, leaving senders to craft messages (sometimes bending the rules) to overcome those flaws.

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.

Receivers that haven't the luxury of strict enforcement of the standards on inbound messages are usually best served by observing the following guidelines for handling of malformed messages:

1. Whenever possible, mitigation of syntactic malformations should be guided by an assessment of the most likely semantic intent. For example, it is reasonable to conclude that multiple sets of angle brackets around an address are simply superflous and can be dropped.

2. When the intent is unclear, or when it is clear but also impractical to change the content to reflect that intent, mitigation should be limited to cases where not taking any corrective action would clearly lead to a worse outcome.

3. Security issues, when present, need to be addressed and may force mitigation strategies that are otherwise suboptimal.

2. Document Conventions

2.1. 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 "message" should be assumed to mean a block of text conforming to the Internet Message Format.

4. Invariant Content

An agent handling a message could use several distinct representations of the message. One is an internal representation, such as separate blocks of storage for the header and body, some header or body alterations, or tables indexed by header name, set up to make particular kinds of processing easier. The other is the representation passed along to the next agent in the handling chain. This might be identical to the message input to the module, or it might have some changes such as added or reordered header fields or body elisions to remove malicious content.

Message handling is usually most effective when each in a sequence of handling modules receives the same content for analysis. A module that "fixes" or otherwise alters the content passed to later modules can prevent the later modules from identifing malicious or other content that exposes the end user to harm. It is important that all processing modules can make consistent assertions about the content. Modules that operate sequentially sometimes add private header fields to relay information downstream for later filters to use (and possibly remove), or they may have out-of-band ways of doing so. Whenever possible, the latter mechanism should be used.

The above is less of a concern when multiple analysis modules are operated in parallel, independent of one another.

Often, abuse reporting systems can act effectively only when a complaint or report contains the original message exactly as it was generated. Messages that have been altered by handling modules might render a complaint inactionable as the system receiving the report may be unable to identify the original message as one of its own.

Some message changes alter syntax without changing semantics. (Indeed, analyzing the semantics of malformations was the impetus for this work.) For example, Section 7.4 describes a situation where an

agent removes additional header whitespace. This is a syntax change without a change in semantics, though some systems (e.g., DKIM) are sensitive to such changes. Message system developers need to aware of the downstream impact of making either kind of change.

There will always be local handling exceptions, but these guidelines should be useful for developing integrated message processing environments.

In most cases, this document only discusses techniques used on internal representations. It is occasionally necessary to make changes between the input and output versions; such cases will be called out explicitly.

5. 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 Handling Service (MHS; see [EMAIL-ARCH]). This is the component that is essentially the interface between end users that create content and the mail stream.

MHSes need to become more strict about enforcement of all relevant email standards, especially [MAIL] and the [MIME] family of documents.

More strict conformance by relaying MTAs will also be helpful. 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.

6. Line Termination

For interoperable Internet Mail messages, the only valid line separation sequence during a typical SMTP session is ASCII 0x0D ("carriage return", or CR) followed by ASCII 0x0A ("line feed", or LF), commonly referred to as CRLF. This is not the case for binary mode SMTP (see [BINARYSMTP]).

Common UNIX user tools, however, typically only use LF for internal line termination. This means that a protocol engine, which converts between UNIX and Internet Mail formats, has to convert between these two end-of-line representations before transmitting a message or after receiving it.

Non-compliant 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] and [MAIL], this means the entire message actually exists on a single line.

Within modern Internet Mail it is highly unlikely that an isolated CR or LF is valid in common ASCII text. Furthermore, [MIME] presents mechanisms for encoding content that actually does need to contain such an unusual character sequence.

Thus, it will typically be safe and helpful to treat a naked CR or LF as equivalent to a CRLF when parsing a message.

Note that this advice pertains only to the raw SMTP data, and not to decoded MIME entities. In fact, it could be necessary to encode MIME content that actually does need to contain such unusual character sequences. However, they would by definition be encoded, and not visible in the raw SMTP stream.

7. Header Anomalies

This section covers common syntactical and semantic anomalies found in a message header, and presents preferred mitigations.

7.1. Converting Obsolete and Invalid Syntaxes

A message using an obsolete header syntax might confound an agent that is attempting to be robust in its handling of syntax variations. A bad actor could exploit such a weakness in order to get abuse or malicious content through a filter. This section presents some examples of such variations. Messages including them ought be rejected; where this is not possible, recommended internal interpretations are provided.

7.1.1. Host-Address Syntax

The following obsolete syntax attempts to specify source routing:

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

This means "send to fran@example.com via the mail service at example.net". It can safely be interpreted as:

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

7.1.2. Excessive Angle Brackets

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

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

can safely be interpreted as:

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

7.1.3. Unbalanced Angle Brackets

The following use of unbalanced angle brackets:

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

can usually be treated as:

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

7.1.4. 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>
```

7.1.5. Commas in Address Lists

This use of an errant comma:

```
To: <third@example.net, fourth@example.net>
```

can reasonably be interpreted as ending an address, so the above should really be interpreted as:

```
To: third@example.net, fourth@example.net
```

7.1.6. Unbalanced Quotes

The following use of unbalanced quotation marks:

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

leaves software with no obvious "good" interpretation. If it is essential to extract an address from the above, one possible interpretation is:

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

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

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

7.1.7. Naked Local-Parts

[MAIL] defines a local-part as the user portion of an email address, and the display-name as the "user-friendly" label that accompanies the address specification.

Some broken submission agents might introduce messages with only a local-part or only a display-name and no properly formed address. For example:

```
To: Joe
```

A submission agent ought to reject this or, at a minimum, append "@" followed by its own host name or some other valid name likely to enable a reply to be delivered to the correct mailbox. Where this is not done, an agent receiving such a message will probably be successful by synthesizing a valid header field for evaluation using the techniques described in Section 7.6.

7.2. Non-Header Lines

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.

7.3. 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.

This example was legal in earlier versions of the Internet Mail format standard.

The best handling of this example is for a message parsing engine to behave as if line {4} was not present in the message and for a message creation engine to emit the message with line {4} removed.

7.4. Header Malformations

Among the many possible malformations, 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. Therefore, it is best to remove all whitespace after the field name but before the colon and to emit the field in this modified form.

7.5. 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 typically 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. Some popular open source filtering programs and some popular Mailing List Management (MLM) packages select either the first or last instance of a particular field name, such as From, to decide who sent a message. Absent strict 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 message validity is assessed, such as through enhanced authentication methods. Such methods might cover one instance of a constrained field but not another, taking the wrong one as "good" or "safe". An MUA, for example could show the first of two From fields to an end user as "good" or "safe" while an authentication method actually only verified the second.

In attempting to counter this exposure, one of the following can be enacted:

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. where a field has a limited instance count, combine additional instances into a single compound instance;

4. where a field can contain multiple distinct values (such as From) or is free-form text (such as Subject), combine them into a semantically identical single header field of the same name (see Section 7.5.1);

5. 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-".

Selecting a mitigation action from the above list, or some other action, must consider the needs of the operator making the decision, and the nature of its user base.

7.5.1. Repeated Header Fields

There are some occasions where repeated fields are encountered where only one is expected. Two examples are presented. First:

```
From: reminders@example.com {1}
To: jqpublic@example.com {2}
Subject: Automatic Meeting Reminder {3}
Subject: 4pm Today -- Staff Meeting {4}
Date: Wed, 20 Oct 2010 08:00:00 -0700 {5}

Reminder of the staff meeting today in the small {6}
auditorium.  Come early! {7}
```

The message above has two Subject fields, which is in violation of Section 3.6 of [MAIL]. A safe interpretation of this would be to treat it as though the two Subject field values were concatenated, so long as they are not identical, such as:

```
From: reminders@example.com {1}
To: jqpublic@example.com {2}
Subject: Automatic Meeting Reminder {3}
  4pm Today -- Staff Meeting {4}
Date: Wed, 20 Oct 2010 08:00:00 -0700 {5}

Reminder of the staff meeting today in the small {6}
auditorium.  Come early! {7}
```

Second:

```
From: president@example.com {1}
From: vice-president@example.com {2}
To: jqpublic@example.com {3}
Subject: A note from the E-Team {4}
Date: Wed, 20 Oct 2010 08:00:00 -0700 {5}

This memo is to remind you of the corporate dress {6}
code.  Attached you will find an updated copy of {7}
the policy. {8}
...
```

As with the first example, there is a violation in terms of the number of instances of the From field. A likely safe interpretation would be to combine these into a comma-separated address list in a single From field:

```
From: president@example.com, {1}
      vice-president@example.com {2}
To: jqpublic@example.com {3}
Subject: A note from the E-Team {4}
Date: Wed, 20 Oct 2010 08:00:00 -0700 {5}

This memo is to remind you of the corporate dress {6}
code.  Attached you will find an updated copy of {7}
the policy. {8}
...
```

7.6. Missing Header Fields

Similar to the previous section, there are messages seen in the wild that lack certain required header fields. In particular, [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 based on data provided in other parts of the protocol

Option 2 is recommended for handling this case. Handling agents should add these for internal handling if they are missing, but should not add them to the external representation. The reason for this advice 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 in a way

visible to other components.

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 representation, since many deployed tools use the content of that field as a common unique message reference, so its absence inhibits correlation of message processing. Section 3.6.4 of [MAIL] describes advisable practise for synthesizing the content of this field when it is absent, and establishes a requirement that it be globally unique.

7.7. Return-Path

A valid message will have exactly one Return-Path header field, as per Section 4.4 of [SMTP]. Should a message be encountered bearing more than one, all but the topmost one is to be disregarded, as it is most likely to have been added nearest to the mailbox that received that message.

7.8. Missing or Incorrect Charset Information

MIME provides the means to include textual material employing charsets other than US-ASCII. Such material is required to have an identifiable charset. Charset identification is done using a "charset" parameter in the Content-Type header field, a character set label within the MIME entity itself, or the character set may be implicitly specified by the Content-Type (see [CHARSET]).

It is unfortunately fairly common for required character set information to be missing or incorrect in textual MIME entities. As such, processing agents should perform basic sanity checks, e.g.:

- US-ASCII is 7bit only, so 8bit material is necessarily not US-ASCII.
- UTF-8 has a very specific syntactic structure that other 8bit charsets are unlikely to follow.
- Null bytes (ASCII 0x00) are not allowed in either 7bit or 8bit data.

- Not all 7bit material is US-ASCII. The presence of the various escape sequences used for character switching may be used as an indication of the various ISO-2022 charsets.

When a character set error is detected, processing agents should:

a. apply heuristics to determine the most likely character set and, if successful, proceed using that information; or

b. refuse to process the malformed MIME entity.

A null byte inside a textual MIME entity 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. Accordingly, null bytes require additional special handling.

A few null bytes in isolation is likely to be the result of poor message construction practices. Such nulls should be silently dropped.

Large numbers of null bytes are usually the result of binary material that is improperly encoded, improperly labeled, or both. Such material is likely to be damaged beyond the hope of recovery, so the best course of action is to refuse to process it.

Finally, the presence of null bytes may be used as indication of possible malicious intent.

7.9. Eight-Bit Data

Standards-compliant email 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.

The best way to handle non-compliant 8bit material depends on its location.

Non-compliant 8bit in MIME entity content should simply be processed as if the necessary SMTP extensions had been used to transfer the message. Note that improperly labeled 8bit material in textual MIME entities may require treatment as described in Section 7.8.

Non-compliant 8bit in message or MIME entity header fields can be handled as follows:

- Occurrences in unstructured text fields, comments, and phrases, can be converted into encoded-words (see [MIME3] if a likely character set can be determined. Alternatively, 8bit characters can be removed or replaced with some other character.

- Occurrences in header fields whose syntax is unknown may be handled by dropping the field entirely or by removing/replacing the 8bit character as described above.

- Occurrences in addresses are especially problematic. Agents supporting [EAI] may, if the 8bit conforms to 8bit syntax, elect to treat the messages as an EAI message and process it accordingly. Otherwise, it is in most cases best to exclude the address from any sort of processing -- which may mean dropping it entirely -- since any attempt to fix it definitively is unlikely to be successful.

8. MIME Anomalies

[MIME], et seq, include 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.

8.1. Missing MIME-Version Field

Any message that uses [MIME] constructs is required to have a MIME-Version header field. Without it, 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. It is prudent to disregard this absence and conduct analysis of the message as if it were present, especially by agents attempting to identify malicious material.

Further, the absence of MIME-Version might be an indication of malicious intent, and extra scrutiny of the message may be warranted. Such omissions are not expected from compliant message generators.

8.2. Faulty Encodings

There have been a few different specifications of base64 in the past. The implementation defined in [MIME] instructs decoders to discard characters that are not part of the base64 alphabet. Other implementations consider an encoded body containing such characters

to be completely invalid. Very early specifications of base64 (see [PEM], for example) allowed email-style comments within base64-encoded data.

The attack vector here involves constructing a base64 body whose meaning varies given different possible decodings. If a security analysis module wishes to be thorough, it should consider scanning the possible outputs of the known decoding dialects in an attempt to anticipate how the MUA will interpret the data.

9. Body Anomalies

9.1. 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.

To ensure long lines do not prevent analysis of potentially malicious data, handling agents are strongly encouraged to 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.

10. Security Considerations

The discussions of the anomalies above and their prescribed solutions are themselves security considerations. The practises enumerated in this document are generally perceived as attempts to resolve security considerations that already exist rather than introducing new ones. However, some of the attacks described here may not have appeared in previous email specifications.

11. IANA Considerations

This document contains no actions for IANA.

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

12. References

12.1. Normative References

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

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

12.2. Informative References

[BINARYSMTP] Vaudreuil, G., "SMTP Service Extensions for Transmission of Large and Binary MIME Messages", RFC 3030, December 2000.

[CHARSET] Melnikov, A. and J. Reschke, "Update to MIME regarding "charset" Parameter Handling in Textual Media Types", RFC 6657, July 2012.

[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.

[EAI] Yang, A., Steele, S., and N. Freed, "Internationalized Email Headers", RFC 6532, February 2012.

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

[MIME3] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

[PEM] Linn, J., "Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures", RFC 1113, August 1989.

[RFC733] Crocker, D., Vittal, J., Pogran, K., and D. Henderson, Jr., "Standard for the Format of Internet Text Messages", RFC 733, November 1977.

[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: Dave Cridland, Dave Crocker, Jim Galvin, Tony Hansen, John Levine, Franck Martin, Alexey Melnikov, and Timo Serainen

Authors' Addresses

Murray S. Kucherawy

EMail: superuser@gmail.com

Gregory N. Shapiro

EMail: gshapiro@sendmail.com

N. Freed

EMail: ned.freed@mrochek.com