RFC 7103 : Advice for Safe Handling of Malformed Messages

Stéphane Bortzmeyer
<stephane+blog@bortzmeyer.org>

Première rédaction de cet article le 16 janvier 2014

Date de publication du RFC : Janvier 2014

https://www.bortzmeyer.org/7103.html

Dans un monde idéal, les messages transmis par le système de courrier électronique de l'Internet seraient tous parfaitement conformes aux normes, comme le RFC 5322[1]. Dans la réalité, l'ancienneté du courrier, et son succès, qui a mené à une explosion du nombre de logiciels, font que beaucoup de messages sont incorrects, « "*malformed*" », dit ce nouveau RFC. Que doivent faire les logiciels confrontés à ces messages incorrects ? Voici une série d'avis pour les programmeurs qui veulent essayer de faire le moins mal possible.

Les normes du courrier électronique sont pourtant anciennes, remontant aux années 1970 (le RFC 733 semble avoir été le premier), et on pourrait penser qu'elles sont désormais bien intégrées partout. Mais ce n'est clairement pas le cas en pratique. Une approche « fasciste » de ce problème serait de rejeter ces messages invalides. Mais cela éliminerait une bonne partie de l'intérêt du courrier électronique, la capacité à communiquer avec un grand nombre de gens. La pression du « marché » va donc plutôt vers un certain laxisme. Un logiciel qui rejetterait tous les messages erronés n'auraient pas beaucoup de clients ! Il est en général plutôt conseillé de faire preuve de souplesse et d'accepter ces messages non-standard dans certains cas, dans la logique du principe de robustesse <https://www.bortzmeyer.org/principe-robustesse.html>. Cela ne veut pas dire qu'il faut tout tolérer et ce RFC 7103 rassemble des conseils pour les auteurs de logiciels de gestion du courrier. Une interprétation extrémiste du principe de robustesse serait en effet qu'un émetteur peut envoyer n'importe quoi, le récepteur ayant tout à faire tout travail d'interprétation (« mais qu'est-ce qu'il a voulu dire ? »).

Une autre variante du problème de non-conformité aux normes est que certains logiciels, côté réception, ont des bogues qui peuvent pousser les émetteurs à violer la norme... pour ne pas déclencher ces bogues. Bref, ce RFC est confus parce que le problème est confus. Le courrier électronique est un gros b....l.

1. Pour voir le RFC de numéro NNN, https://www.ietf.org/rfc/rfcNNN.txt, par exemple https://www.ietf.org/rfc/rfc5322.txt

Le principe de robustesse <https://www.bortzmeyer.org/principe-robustesse.html> a plutôt bien marché pour le courrier, estime notre RFC. Faire un message qui sera accepté partout est relativement facile, vu la tolérance des récepteurs face à des erreurs de détail. Mais les temps ont changé : notamment, la montée des préoccupations de sécurité rend difficile de continuer à être tolérant. Par exemple, un méchant peut placer un piège dans un message invalide, piège qui sera détecté par certains logiciels et pas par d'autres. Cette différence d'appréciation, inoffensive pour des messages normaux, peut devenir critique face à des messages portant du logiciel malveillant.

Un autre cas où le problème peut devenir critique est celui du spam. Les logiciels d'envoi de spam sont souvent particulièrement négligents (vite et mal écrits) et pas mal de logiciels antispam mettent des mauvais points aux messages qui ont des erreurs techniques, ce qui peut mener à leur classement comme spam. Le principe de robustesse rencontre donc une autre limite ici.

Il faut se rappeler que le courrier est non-interactif, ce qui rend difficile de faire respecter les règles. Dans une communication interactive, on peut imaginer un récepteur qui rejette les messages invalides, indiquant clairement à l'émetteur qu'il doit recommencer en respectant les règles. Pour le courrier, ce n'est pas possible : le récepteur doit se débrouiller avec ce qu'il a. L'auteur du logiciel de réception préférera donc toujours accepter les messages invalides plutôt que de subir les plaintes de ses utilisateurs, qui ne recevraient pas les messages attendus.

Néanmoins, notre RFC précise (section 1.2) qu'il n'approuve pas ces innombrables erreurs que commettent les émetteurs négligents. Le code en cause devrait être réparé de toute façon, ne serait-ce que pour des raisons de sécurité. Mais les conseils donnés par la suite sont là pour permettre au courrier de fonctionner en attendant cette réparation (fort hypothétique, je dois dire). Cela n'empêche pas de rappeler aux émetteurs de courrier qu'il est dans leur intérêt, et dans celui de tout l'Internet, d'essayer de fabriquer et d'envoyer des messages le plus parfaits possible. Bien relire les RFC 5598, RFC 5322 et RFC 2045 est donc nécessaire si on écrit du logiciel qui traite des messages.

Pour les récepteurs des messages mal formés, notre RFC suggère les principes suivants :

— Lorsqu'il y a une erreur syntaxique mais que l'intention sémantique est claire (exemple : un excès de ¡ autour d'une adresse), accepter le message.
— Autrement, ne pas essayer d'accepter le message, sauf si on a de bonnes raisons de penser qu'un rejet aurait de pires conséquences.
— Et ne pas oublier la sécurité, qui peut imposer des choix qui ne sont pas conformes aux deux principes ci-dessus.

Maintenant, les conseils pratiques. D'abord, en section 4, tout ce qui est lié à la différence entre fond et forme : un message électronique peut avoir plusieurs représentations équivalentes du point de vue du contenu. C'est déjà le cas en transit mais c'est encore plus vrai lorsqu'il est stocké sur un système. Il a pu, par exemple, être mis dans un SGBD. Le principe que pose notre RFC est qu'il faut toujours garder quelque part la forme originale et que c'est elle qui doit être passée aux différents modules qui vont traiter le message. C'est la seule façon de garantir que chaque module ait toute l'information disponible, qu'on ne l'ait pas digérée pour lui. Par exemple, si un module ajoute des en-têtes locaux au message, les modules ultérieurs doivent recevoir le message sans ces en-têtes (si on veut communiquer avec eux, il vaut mieux utiliser des méthodes extérieures au message).

Cet avis est particulièrement important pour les modules voués au traitement de messages abusifs, que l'on souhaite signaler : les modifications faites par les modules précédents, même dans une bonne intention, pourraient empêcher, par exemple, de reconnaître le vrai expéditeur.

Même des changements qui semblent purement syntaxiques (supprimer les espaces non significatifs, par exemple), peuvent affecter d'autres traitements (comme les signatures).

Mais, au fait, quel agent dans l'ensemble de la chaîne de traitement a le plus de responsabilité dans la production de messages corrects ? La section 5 de notre RFC confie ce rôle au MSA : premier jalon en dehors de la machine de l'utilisateur, il est le point recommandé pour le respect des règles. Notamment, il a encore la possibilité de rejeter un message de façon à ce que le vrai émetteur soit prévenu, ce que ne pourront pas toujours faire les MTA situés plus loin sur le trajet (qui peuvent générer un courrier de rejet mais ne sont jamais sûrs qu'il ira à l'émetteur original).

Une cause fréquente de non-conformité des messages est une fin de ligne incorrecte (section 6). La seule fin de ligne standard dans un message est la combinaison CR LF ("*Carriage Return*", 13 en ASCII suivi de "*Line Feed*", 10 en ASCII), sauf si on utilise le RFC 3030. Mais d'autres systèmes ont des conventions différentes, par exemple Unix, où la fin de ligne est le seul LF. En pratique, on voit sur le réseau des messages avec uniquement des LF, et des messages avec un mélange de LF et de CR LF !

Comme il y a très peu de chance qu'un CR ou un LF isolé se retrouvent dans un texte, notre RFC recommande de les traiter comme des CR LF et donc des fins de ligne. (Cet avis ne vaut que pour le texte transporté en SMTP brut, pas pour des parties encodées en MIME.)

Une grosse section du RFC, la 7, est ensuite consacrée à toutes les erreurs qu'on peut trouver dans les en-têtes d'un message. Ce Musée des Horreurs doit son existence en partie à la longévité du courrier électronique, utilisé et maltraité partout depuis de nombreuses années. Ainsi, cet en-tête :

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

est marqué comme dépassé par le RFC 5322 (c'était du routage par la source : envoyer d'abord à `example.net`, pour qu'il transmette à `fran@example.com`). Mais l'intention est claire et on peut donc l'interpréter sans risques comme :

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

Même chose pour les excès de chevrons déjà cités. Il n'y a pas de mal à traiter :

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

comme étant en fait :

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

C'est plus gênant quand ces chevrons ne sont pas égaux en ouverture et en fermeture :

```
To: <another@example.net
```

peut quand même être lu :

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

Et si ce sont les guillemets qui ne sont pas équilibrés ?

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

Ce cas est plus vicieux car il y a plusieurs interprétatations possibles. Mais le RFC recommande la plus simple :

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

Autre bogue qui se voit dans la nature, les lignes dans l'en-tête qui ne sont pas des en-têtes :

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

Manifestement, le sujet, trop long, a subi un retour chariot anormal : les en-têtes sur plusieurs lignes sont légaux dans le format des messages, mais les lignes de continuation doivent commencer par un espace (section 2.2.3 du RFC 5322). Les conséquences de cette bogue peuvent être graves : certains logiciels vont considérer que les en-têtes s'arrêtent à la première ligne qui n'est pas un en-tête, et donc ne verront pas le champ `Date:`, repoussé dans le corps du message, d'autres vont corriger en reconnaissant la ligne anormale et d'autres enfin rejetteront complètement le message comme étant invalide (ce qu'il est). Ce genre de différences de comportement peut être exploité par un attaquant : s'il sait que l'antispam a un comportement, et le MUA un autre, il peut glisser des informations qui échapperont à l'antispam mais seront vues par le lecteur. Le RFC suggère donc, mais "*mezzo voce*", de rejeter le message.

Il y a aussi des cas où les normes n'étaient pas claires, menant à des différences de comportement des logiciels. Ce message était parfaitement légal :

```
From: user@example.com
To: userpal@example.net
Subject: This is your reminder

 about the football game tonight
Date: Wed, 20 Oct 2010 20:53:35 -0400
```

Le champ `Subject:` a été replié en trois lignes, chacune commençant bien par un espace. Donc, ce message n'a normalement qu'une seule interprétation possible (« "*This is your reminder about the football game tonight*" »). Mais certains logiciels vont d'abord supprimer les espaces inutiles, la seconde ligne du sujet va alors être vide, devenant un séparateur entre l'en-tête et le corps... Depuis le RFC 5322, ce message est toujours légal mais marqué comme utilisant une syntaxe dépassée. Aujourd'hui, les émetteurs ne devraient plus envoyer de telles lignes.

On voit aussi des en-têtes ayant diverses erreurs de syntaxe :

```
MIME-Version : 1.0
```

Ici, il y a un espace anormal, entre le nom du champ et le :. Comme l'interprétation est évidente, le RFC recommande de supprimer les espaces avant le séparateur « : ».

Autre cas rigolo, le nombre d'en-têtes. Certains en-têtes ont une cardinalité imposée (voir le tableau au début de la section 3.6 du RFC 5322). C'est ainsi que `From:` et `Subject:`, s'ils sont présents, doivent l'être à un seul exemplaire, alors que `Received:`, en-tête de trace, peut être répété. Que doit faire un logiciel lorsqu'il rencontre :

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

Reminder of the staff meeting today in the small
auditorium.  Come early!
```

Certains logiciels ne vont garder que le premier sujet, d'autres que le dernier. Là encore, un attaquant peut se dissimuler à certains logiciels en fabriquant des messages invalides, qui seront interprétés de manière différente par les différents logiciels. Pour le sujet, le RFC recommande tout simplement de concaténer les deux valeurs (« "*Automatic Meeting Reminder 4pm Today – Staff Meeting*" ») en un seul sujet. Même chose pour l'expéditeur (on l'oublie souvent mais un champ `From:` peut parfaitement contenir plusieurs adresses). Donc,

```
From: president@example.com
From: vice-president@example.com
```

devient :

```
From: president@example.com, vice-president@example.com
```

Et si c'est le contraire ? Si des champs obligatoires dans l'en-tête comme `From:`, `Message-ID:` ou `Date:` sont manquants ? Le choix recommandé par notre RFC est de les ajouter, en synthétisant l'information (pour `Date:`, c'est assez facile, pour `From:`, une solution est d'extraire l'adresse du `MAIL FROM` de SMTP, une autre est de fabriquer à partir de l'adresse IP du client, par exemple `unknown@[192.0.2.2]`). Mais attention : rappelez-vous la régle précédente comme quoi les modules de traitement doivent recevoir le message original. Par exemple, un module antispam peut être intéressé par le fait que `Date:` manque, cela peut être une signature révelant un problème de spam.

Et les problèmes liés au jeu de caractères et à l'encodage ? Les erreurs d'étiquetage sont fréquentes, par exemple un message marqué :

```
Content-Type: text/plain; charset=US-ASCII
```

peut parfaitement contenir des caractères en dehors d'ASCII. Pour détecter l'encodage, notre RFC suggère des heuristiques simples (« si aucun octet n'a le bit de poids fort à zéro, c'est probablement de l'ASCII, sauf si on trouve les séquences d'échappement d'encodages connus comme ISO-2022-JP ; sinon, si le texte est de l'UTF-8 légal, c'est probablement de l'UTF-8 car cela ne peut pas arriver par accident »). Un logiciel de courrier est donc autorisé à utiliser ces heuristiques pour trouver le vrai jeu de caractères et l'encodage utilisé, afin de traiter ce message selon ce jeu et pas selon celui indiqué dans les en-têtes.

Puisqu'on en vient aux jeux de caractères, il faut noter que l'introduction de MIME a suscité de nouvelles occasions de bogues, au point qu'une section entière de notre RFC, la 8, leur est consacrée. Par exemple, Base64 a fait l'objet de plusieurs spécifications (la « bonne » est aujourd'hui celle du RFC 4648) et toutes n'étaient pas d'accord entre elles sur l'attitude à adopter lors de la rencontre de caractères situés en dehors de l'alphabet limité de Base64. Là encore, comme tout ce qui permet deux interprétations, cela peut être utilisé par un attaquant pour produire un message qui sera ignoré par l'IDS ou l'antispam mais accepté par le lecteur.

Enfin, le dernier problème envisagé dans ce RFC est celui des lignes trop longues : la norme (section 2.1.1 du RFC 5322) les limite à 1 000 caractères mais on trouve parfois davantage. Le RFC ne donne pas de conseil explicite (rejeter ? corriger ?) mais met en garde contre les logiciels qui ignorent les caractères supplémentaires : cela fournit encore un autre moyen de dissimulation de contenu.

À noter qu'une des propositions alternatives, lors de la création de ce RFC, était de faire un registre en ligne, contenant les erreurs dans les messages et les actions conseillées. Cela aurait été plus facile à mettre à jour qu'un RFC. Mais documenter dans un registre des erreurs semblait trop étrange...