Lutte anti-spam concrète et pratique avec du logiciel libre

Stéphane Bortzmeyer
AFNIC
bortzmeyer@nic.fr

Pierre David
Centre Réseau Communication, Université Louis Pasteur, Strasbourg
Pierre.David@crc.u-strasbg.fr

Résumé

Le spam, par la consommation de ressources matérielles et humaines qu'il nécessite, par le temps qu'il fait perdre à ses lecteurs involontaires, par l'agacement ou les craintes qu'il suscite est un des principaux fléaux de l'Internet aujourd'hui.

Il existe d'innombrables colloques et commissions sur le spam, et ce depuis des années. Il y a également beaucoup de produits commerciaux payants pour le filtrer, la plupart n'expliquant pas du tout leur fonctionnement et donc leurs limites ou leurs risques.

*Mais il existe peu de documentations **pratiques** et **globales** sur la lutte anti-spam telle que doit la pratiquer l'ingénieur système moyen, qui ne peut pas attendre la signature d'un Traité International ou le résultat d'un procès, et qui doit arrêter le flot aujourd'hui.*

*Chaque logiciel anti-spam libre, et il y en a beaucoup, vient avec sa documentation mais les documentations transversales, sur la combinaison de ces logiciels, sur leur choix, sont beaucoup plus rares. Etant donné qu'il est largement acquis que la lutte contre le spam ne peut pas reposer sur une seule technique, ce tutoriel va tenter de présenter l'installation et la configuration d'un ensemble de techniques efficaces pour lutter contre le spam **entrant**.*

Il ne tentera pas de faire le tour de toutes les méthodes, seulement de celles qui sont raisonnables pour l'administrateur système d'aujourd'hui. Cette tâche de filtrage des solutions est une composante importante de ce tutoriel car la lutte contre le spam comprend plus de rebouteux que d'experts.

Toutes les techniques seront illustrées par des exemples concrets avec les serveurs Postfix et Sendmail, les deux logiciels libres de loin les plus répandus pour gérer le courrier.

Le tutoriel s'adresse à des administrateurs système en charge d'un serveur de messagerie. Une compétence minimum en configuration de Postfix ou bien de Sendmail est nécessaire.

Mots clefs

spam, greylisting, SpamAssassin, bogofilter, listes noires, postfix, Sendmail

1 Introduction

Si vous croyez que le *spam* dont nous parlons est une marque de pseudo-viande en boîte[1], il est préférable de lire d'abord [1]. Pour les actions des pouvoirs publics, voir [2] ou [3]. Pour les exposés passés à JRES, voir [4]. Sinon, entrons dans le vif du sujet.

Donc, vous êtes administrateur d'un ou plusieurs serveurs de messagerie, vos utilisateurs reçoivent plus de *spam* que de courrier légitime, vous voulez faire quelque chose, et vous n'êtes pas un gourou SMTP, vous n'avez jamais lu le code de Postfix, vous ne connaissez pas par cœur le RFC 2822[2], et, surtout, vous n'avez pas le temps de faire tout cela.

Vous avez déjà vu passer plein de publicités[3] pour des produits commerciaux anti-*spam* dont vous soupçonnez qu'ils sont largement placebo, et vous avez le tournis devant la quantité de solutions proposées.

Avant de lister l'avis des spécialistes[4], quelques révisions seront utiles.

[1] http://www.spam.com/
[2] Pour voir le RFC de numéro NNN, http://www.ietf.org/rfc/rfcNNN.txt, par exemple http://www.ietf.org/rfc/rfc2822.txt
[3] Souvent envoyées par *spam* . . .
[4] Outre ce tutoriel, on peut aussi consulter [5] pour avoir tous les liens utiles.

1.1 Rappel rapide de l'architecture du courrier

Ce tutoriel suppose une connaissance de base de SMTP. Nous signalons juste les points essentiels pour la lutte anti-*spam*.

Le courrier est écrit et lu par des humains, qui se servent pour cela de MUA[5]. Il est transmis ensuite[6] du premier MUA à un MTA[7] et il ira ensuite de MTA en MTA jusqu'à destination. Là, il sera déposé dans la boîte aux lettres de l'utilisateur par un MDA[8].

Le protocole utilisé pour le transport est SMTP[9], spécifié dans le RFC 2821. Les informations transportées par SMTP (comme l'adresse à laquelle doivent être envoyés les avis de non-remise) forment l'**enveloppe**.

Le message est structuré selon le RFC 2822. Il comporte deux parties, les en-têtes et le corps (lui-même structuré par la norme MIME[10], RFC 2045).

Les en-têtes sont de la forme {nom, valeur}. Certains sont plutôt destinés à être affichés aux humains comme `Date` qui identifie la date à laquelle a été écrite un message[11] ou bien plutôt destinés aux examens détaillés par l'expert, ce qui est notamment le cas des en-têtes `Received` qui identifient les relais par lesquels est passé le message. Voici un jeu d'en-têtes typique :

```
Received: from maya.nic.fr [192.134.4.160]
        by localhost with POP3 (fetchmail-6.2.5)
        for bortzmeyer@localhost (single-drop); Fri, 07 Oct 2005 17:19:52 +0200 (CEST)
Received: from mx1.nic.fr (mx1.nic.fr [192.134.4.10])
        by maya40.nic.fr (8.12.4/8.12.4) with ESMTP id j97FJcYa697727
        for <bortzmeyer@nic.fr>; Fri, 7 Oct 2005 17:19:38 +0200 (CEST)
Received: from smtp1-g19.free.fr (smtp1-g19.free.fr [212.27.42.27])
        by mx1.nic.fr (Postfix) with ESMTP id 0B6CD94007
        for <bortzmeyer@nic.fr>; Fri,  7 Oct 2005 17:19:33 +0200 (CEST)
Received: from vagabond.noos.fr (rob67-1-82-231-68-196.fbx.proxad.net [82.231.68.196])
        by smtp1-g19.free.fr (Postfix) with ESMTP id D8C2B4D655
        for <bortzmeyer@nic.fr>; Fri,  7 Oct 2005 17:19:32 +0200 (CEST)
Received: (from pda@localhost)
        by vagabond.ma.maison (8.13.4/8.13.4/Submit) id j97FJUhC042692
        for bortzmeyer@nic.fr; Fri, 7 Oct 2005 17:19:30 +0200 (CEST)
        (envelope-from pda)
Date: Fri, 7 Oct 2005 17:19:30 +0200
To: Stephane Bortzmeyer <bortzmeyer@nic.fr>
Message-ID: <20051007151930.GH42064@vagabond.ma.maison>
Content-Type: text/plain; charset=iso-8859-15
Content-Transfer-Encoding: 8bit
In-Reply-To: <20051007144853.GA15829@nic.fr>
User-Agent: Mutt/1.5.11
X-Bogosity: Ham, tests=bogofilter, spamicity=0.000000, version=0.94.4
Subject: Re: [JRES Tutoriel spam] Relire et ajouter du sendmail
From: Pierre DAVID <Pierre.David@crc.u-strasbg.fr>
```

Le courrier n'est pas forcément transporté directement du serveur émetteur au serveur récepteur. Des relais intermédiaires sont présents, ce qui peut compliquer tout test fait sur l'adresse IP du client SMTP.

Il n'y a pas d'authentification dans SMTP. Cela n'est pas la résultat d'une forte myopie chez ses concepteurs, ni celui d'une confiance absolue dans l'humanité. C'est simplement la constatation d'un fait : il n'existe dans l'Internet aucune structure sur laquelle s'appuyer pour faire de l'authentification. Personne ne délivre de cartes d'identité ou de passeport sur Internet. SMTP ne peut donc pas vérifier des identités qui n'existent pas, qui ne sont pas gérées.

Conséquence du point précédent, rien dans le message n'est garanti authentique. Les en-têtes `Received`, par exemple, ont pu être écrits par le *spammeur* lui-même.

Les normes techniques sont constituées de deux RFC, le RFC 2821 qui normalise le protocole SMTP et le RFC 2822 qui normalise le format des messages, notamment de leurs en-têtes. Ces deux RFC sont loin de traiter tous les problèmes, notamment parce que l'architecture du courrier, et le nom des acteurs et des actions n'est pas normalisé. Certains efforts à l'IETF

[5] *Messsage User Agent*, comme mutt, Thunderbird, Kmail ou Evolution

[6] C'est une légère simplification : il est normalement transmis à un MSA (*Message Submission Agent*) mais, en pratique, ce sont les mêmes logiciels qui sont MSA et MTA.

[7] *Message Transport Agent*, comme Sendmail, courier ou postfix

[8] *Message Delivery Agent*) comme procmail `http://www.procmail.org/` ou maildrop `http://www.courier-mta.org/maildrop/`.

[9] *Simple Mail Transport Protocol*

[10] *Multipurpose Internet Mail Extensions*

[11] Comme toutes les informations contenues dans le message, il ne faut pas lui faire une coinfiance exagérée. Par exemple, l'horloge de la machine émettrice n'est pas forcément synchronisée par NTP.

tentent de combler ce vide, qui fait que même un terme comme *forwarding* n'a pas de définition précise, ce qui rend difficile de spécifier de nouveaux protocoles, ou de prédire les conséquences techniques de nouvelles pratiques.

Enfin, il existe de nombreux scénarios d'usage (*use cases*) et toute nouvelle technique doit, autant que possible, s'assurer qu'elle fonctionne avec la majorité de ces scénarios. Le cas où `alice@wanadoo.fr` écrit de chez elle à `bob@free.fr` n'est pas le seul : il y a aussi les listes de diffusion, les « alias », le *forwarding*, les sites Web avec un lien « Envoyez ceci à un ami », ...

1.2 Comment fonctionne techniquement le spam

Le *spam* est aujourd'hui une industrie [6]. L'époque où les *spammeurs* utilisaient des outils standard, pilotés à la main, est bien révolue. Aujourd'hui, le *spam hardcore* est surtout envoyé par des **zombies**, des ordinateurs MS-Windows infectés par un virus ou un ver et qui obéissent aux ordres de leur commandant. Ces zombies travaillent de concert, au sein de grands réseaux, les *botnets* [12].

Certaines mesures anti-*spam* n'ont donc plus de sens : par exemple, imposer un petit calcul avant tout envoi de courrier ne ralentit pas le *spammeur*, qui a des dizaines de milliers de machines à sa disposition, et ne les paie pas.

Les zombies utilisent du logiciel spécialement adapté, qui viole souvent les RFC. Cette caractéristique est exploitée par exemple par le *greylisting*.

En utilisant les zombies, les *spammeurs* se cachent derrière l'ordinateur d'un tiers, ce qui empêche dans la majorité des cas de remonter jusqu'à la véritable origine du message.

1.3 Concepts en sécurité

La lutte contre le *spam* n'est qu'un aspect de la sécurité informatique en général. De nombreuses leçons apprises dans d'autres domaines n'ont pas encore été appliquées à la lutte anti-*spam*.

D'abord, il n'y a pas de solution parfaite au *spam*, pas plus qu'aux incendies ou aux agressions criminelles. Il faut toujours faire une analyse coût / bénéfice, où on essaie de mettre en rapport les avantages d'une solution avec ses inconvénients [8] [9]. Au lieu de cela, on voit parfois des affirmations ridicules comme ce rapport officiel présenté à un grand organisme international et qui disait qu'en adoptant la méthode X, on supprimait 95 % du *spam*. On aurait aussi bien pu dire qu'en éteignant tous les ordinateurs, on supprimait 100 % du *spam* ! N'annoncer que le gain et oublier le coût n'a aucun intérêt pour évaluer une technique.

Ensuite, aucune solution technique ne résoudra ce problème complexe et où l'ennemi n'est pas une force aveugle mais un groupe de délinquants actifs et compétents[13].

Les solutions de filtrage, qui font l'objet de ce tutoriel, ont toutes des avantages et des inconvénients. Nous avons écarté celles dont les inconvénients peuvent être tellement graves qu'ils suppriment l'intérêt du courrier électronique, ce qui fait qu'ils ne valent pas mieux que le *spam*.

Les principaux défauts des solutions de filtrage sont mesurés par :

– le taux de faux positifs, qui est le pourcentage de messages légitimes, classés à tort comme *spam* ;
– et le taux de faux négatifs, qui est le pourcentage de *spams* classés à tort comme légitimes.

En général, le seul taux de faux positifs est zéro ou en tout cas ϵ, si on veut être réaliste. Par contre, on peut accepter un certain nombre de faux négatifs.

En outre, ces solutions ont un coût. Le temps d'administrateur système en est la composant la plus visible, mais il y en a d'autres : le nombre de machines nécessaire (l'époque où un bête PC pouvait servir toute une université est révolue, aujourd'hui, la plupart des campus déploient une ferme de plusieurs multiprocesseurs juste pour faire tourner SpamAssassin) et la charge sur le réseau (SPF nécessite davantage de requêtes au DNS, le *greylisting* nécessite de traiter davantage de connexions SMTP, etc.).

D'ores et déjà, le *spam* élimine les petits prestataires de messagerie, qui ont du mal à suivre la course aux armements.

[12] Un excellent et passionnant article sur la vie des zombies est [7]. J'en ai apprécié la formule « *A botnet is comparable to compulsory military service for Windows boxes* » (Stromberg)

[13] Le dégonflement de la bulle Internet a laissé un certain nombre d'ingénieurs compétents disponibles pour des activités moins légales mais qui, elles, rapportent et pas du vent.

2 Les techniques de lutte

2.1 Déploiement

Quelques mots sur les techniques de déploiement des différents mécanismes que nous allons voir.

Nous allons surtout parler des MTA mais il faut noter que beaucoup de mécanismes peuvent être mis en œuvre par le MDA. Par exemple, si le MDA est procmail, l'administrateur système peut configurer un fichier commun, typiquement `/etc/procmailrc`, où il mettra des règles communes à tout ses utilisateurs. Cela peut-être plus simple et plus robuste que de configurer le MTA (mais cela ne fonctionne que si on est sur la machine de dépôt du courrier, pas sur un relais intermédiaire).

2.1.1 Postfix

Postfix se contrôle essentiellement par deux fichiers

Le `master.cf` liste les différents processus de Postfix, et leur mécanisme de communication. Postfix peut aussi communiquer avec des programmes extérieurs. En outre, comme init ou inetd, Postfix peut aussi lancer de tels programmes s'il en a besoin.

Le `main.cf`, lui, configure les différentes options de Postfix.

Parmi les plus importantes[14], `smtpd_X_restrictions`, où X vaut `helo` ou `sender` ou `recipient`. Cette option permet de définir un certain nombre de tests lors de la session SMTP. Il est important de noter que l'ordre de ces tests compte : dès qu'un test est positif, Postfix arrête les vérifications. Ainsi, on met souvent en début de cette liste `permit_mynetworks` qui autorise à peu près tout de la part des clients SMTP dont l'adresse IP figure dans la variable `mynetworks`.

Voici des restrictions typiques :

```
# Le fichier badmxcidr contient une liste d'émetteurs "impossibles" (MX invalide)
smtpd_sender_restrictions = reject_unknown_sender_domain,
                    check_sender_mx_access cidr:/etc/postfix/badmxcidr
smtpd_recipient_restrictions =
   permit_mynetworks,
   reject_unauth_destination
```

2.1.2 Sendmail

L'API Milter, disponible depuis la version 8.10 de Sendmail, constitue le cœur des outils de lutte contre le *Spam*. En effet, ce mécanisme permet d'écrire des programmes externes, les *mail filters* (ou *milters*) afin d'exercer une action sur tout message SMTP entrant. La présence d'une bibliothèque de fonction relativement simple explique la floraison[15] de *milters* disponibles.

La figure 1 décrit la communication entre Sendmail et un démon externe : lors de la réception d'un message, Sendmail agissant comme un serveur SMTP transmet, à chaque étape de l'échange SMTP avec le client, des informations via une *socket* [16] à un programme externe. Ce programme reçoit les informations, effectue des traitements appropriés (anti-virus, interrogation SPF, etc.) et renvoie via la même *socket* un code à Sendmail, indiquant entre autres si le courrier doit être jeté, rejeté ou accepté.

[14] Comme le `main.cf` peut être assez long, notons qu'on peut regarder uniquement les paramètres qui n'ont pas leur valeur par défaut avec la commande `postconf -n`

[15] Le site `http://www.milter.org/` en répertorie bon nombre.

[16] Prise permettant de se connecter à un service distant mais l'anglicisme *socket* est très répandu.

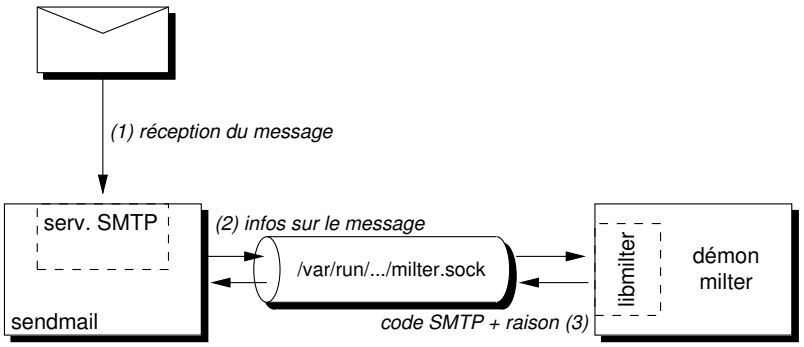

Figure 1 – *Communication entre Sendmail et un milter*

L'écriture d'un démon est grandement facilitée par la bibliothèque de fonctions (`libmilter.a`) ; cette bibliothèque masque tous les détails du protocole utilisé sur la *socket* de communication avec Sendmail, ainsi que la gestion des *threads* qui permettent à une seule instance du programme de gérer les appels simultanés de plusieurs Sendmail répondant à des requêtes SMTP. Dans la pratique, écrire un *milter* se résume à écrire un programme en C pour spécifier les paramètres tels que la localisation de la *socket* (Unix ou TCP), ainsi que des fonctions qui seront appelées à chaque étape (HELO, MAIL FROM, etc.) de la communication entre le client et le serveur SMTP. Pour être complet, signalons qu'il est possible également d'écrire des *milters* dans d'autres langages, comme par exemple Perl grâce à des modules CPAN comme `Sendmail::Milter` ou `Mail::Milter`.

Un *milter* est exécuté dans le contexte d'un processus indépendant de Sendmail. Il est donc démarré séparément, et de préférence avant Sendmail lui-même.

Sendmail peut faire appel à un *milter* distant, la *socket* n'étant pas forcément restreinte à une communication locale. Cela permet des configuration élaborées faisant de la répartition de charge en distribuant les différents *milters* sur des fermes de serveurs, ce qui peut être utile pour des filtres lourds, comme par exemple SpamAssassin.

La présence d'un *milter* est indiquée dans le fichier de configuration de Sendmail (`sendmail.cf`) par les lignes suivantes :

```
#
# Définition des informations envoyées aux milters
#
O Milter.macros.connect=b,j,{if_addr},{daemon_name},{if_name}
O Milter.macros.envfrom=i
O Milter.macros.envrcpt={rcpt_addr}

#
# Définition de la communication avec les milters
#
Xtoto,Socket=local:/var/run/toto/toto.sock, Flags=T, Timeouts=S:4m;R:4m
Xtiti,Socket=local:/var/run/titi/titi.sock, Flags=T, Timeouts=C:15m;S:4m;R:4m;E:10m

#
# Activation des milters
#
O InputMailFilters=toto,titi
```

Ces lignes peuvent être placées directement dans le fichier `sendmail.cf`. Si vous utilisez le Kit Jussieu [10] pour générer votre `sendmail.cf`, vous pouvez placer ces lignes dans un fichier d'inclusion[17]

- les trois premières lignes « O » spécifient les informations à passer aux *milters* lors de chaque étape du dialogue SMTP. Ces informations prennent la forme de *macros*, et devraient être normalement documentées par le fournisseur du *milter*. Au cas où vous en avez plusieurs, spécifiez l'union des informations requises par chaque *milter* ;
- les deux lignes « X » déclarent les *milters* :
  - nom (qui sera utilisé dans la ligne d'activation et dans les logs) ;
  - `Socket=` : emplacement de la *socket*, qui peut être locale ou distante (`inet6:1234@host`) ;
  - `Flags=` : *flags* précisant le comportement de Sendmail si le *milter* est indisponible (`R` pour un rejet de connexion, ou `T` pour un refus temporaire) ;

[17] Sendmail peut aussi se configurer par les macros M4, documentées en `http://www.sendmail.org/m4/`, et qui ne sont pas traitées ici.

  - `Timeouts=` : *timeouts* de connexion au *milter* (`C`), d'envoi d'information (`S`), de lecture de réponse (`R`), ou entre la fin du message et l'acquitement (`E`).
- enfin, la dernière ligne « O » active les *milters*, dans l'ordre spécifié.

Le mécanisme des *milters* est donc très souple et hautement configurable. Il permet d'interagir avec Sendmail sans pour autant ajouter de complexité à ce programme. L'administrateur peut dès lors choisir dans une riche palette d'outils.

Parmi ceux-ci, il faut signaler le milter « j-chkmail[18] ». Malgré tout le bien qu'en pensent les auteurs, celui-ci n'a pas été retenu dans le cadre de ce tutoriel car il ne correspond pas strictement à la définition d'un logiciel libre, il couvre un spectre différent des outils présentés ici, et une présentation sera effectuée par son auteur lors de ces JRES. Il n'en reste pas moins qu'il s'agit d'une solution intéressante, sans équivalent pour d'autres MTA que Sendmail, et qui peut avantageusement être vue comme un complément des techniques présentées ici.

Les *milters* sont des programmes additionnels, il faut donc en assurer l'installation et le suivi. L'utilisation d'un système Unix qui facilite cette tâche est vivement recommandée. Par exemple, tous les *milters* listés dans ce tutoriel sont disponibles dans les « ports » de FreeBSD, et l'installation de ces outils se résume à un appel au programme `portinstall`. De même, le fait de disposer des versions les plus à jour repose sur la mise à jour de l'« arbre des ports » et l'utilisation du programme `portupgrade`. D'autres systèmes disposent d'outils comparables (par exemple, avec Debian, `apt-cache search milter` pour trouver le nom des *milters* et `apt-get install $MILTER` pour l'installer) aussi nous ne décrirons donc pas ici les procédures d'installation.

2.2 *greylisting*

Le « *greylisting* » correspond à l'envoi délibéré d'une erreur SMTP 4xx (erreur temporaire, alors que les 5xx sont définitives, voir RFC 2821) lorsqu'on voit un nouvel émetteur. Celui-ci, s'il est un MTA normal, réessaiera plus tard (typiquement un quart d'heure après) et son message sera alors accepté. La plupart des logiciels de *spam* ne réessaieront jamais, car pour envoyer un message à des millions de destinataires, cela coûte trop cher de maintenir une file d'attente pour la réémission des messages.

Tout nouvelle tentative, identifiée par le triplet <adresse IP source, adresse d'enveloppe pour l'émetteur, adresse d'enveloppe pour le destinataire>, est mise dans la « liste grise » ce qui signifie son rejet temporaire. Au bout d'un délai paramétrable (quelques minutes suffisent), le triplet est mis dans la « liste blanche » pour une durée là aussi paramétrable (quelques jours). Toute nouvelle tentative avec ce triplet passera sans délai, et prolongera d'autant le délai de sortie de la « liste blanche ».

Cette technique [11] est très efficace et bloque tous les *spams* qui ne sont pas transmis par un relais ouvert ou par le MTA d'un prestataire.

Elle a quelques rares faux positifs, dûs à des serveurs légitimes mais bogués car ils ne gèrent pas de file d'attente ou bien ils interprètent les codes SMTP 4xx comme des erreurs définitives. Le résultat est qu'ils ne tentent pas de réémission et on utilise donc en général le *greylisting* en conjonction avec une liste blanche.

Cette technique est recommandée.

Elle est simple à mettre en œuvre mais son avenir est incertain : si tout le monde l'adopte, les *spammeurs* commenceront à réemettre, ce qu'ils ne font pas aujourd'hui. Au moins, cela donnera leur chance à des techniques comme les filtres à empreintes.

2.2.1 *greylisting* avec Postfix

La meilleure solution est d'utiliser le *policy filter*, déjà présenté. Le stockage des triplets est réalisé par Postgrey[19].

On n'utilise pas `master.cf`, car le serveur Postgrey a besoin de tourner en permanence.

On configure Postfix, dans `main.cf`, pour soumettre les sessions SMTP au test :

```
smtpd_recipient_restrictions = permit_mynetworks, ...
   check_policy_service inet:127.0.0.1:60000, ...
```

Ici, Postgrey va écouter sur le port 60000 et renvoyer à Postfix sa décision.

[18] `http://j-chkmail.ensmp.fr/`
[19] `http://isg.ee.ethz.ch/tools/postgrey/`

2.2.2 *greylisting* avec Sendmail

Le support du *greylisting* est fourni par plusieurs *milters*. Celui que nous utilisons est le très puissant et très efficace « milter-greylist[20] » (version $\geq$ 2.0) d'Emmanuel Dreyfus. Les caractéristiques de ce *milter* sont :

- support d'IPv6 ;
- support de SPF ;
- support des utilisateurs nomades à travers l'authentification SMTP ;
- support de listes d'accès pour des adresses IP ou des adresses électroniques spécifiques ;
- maintien des listes grise et blanche en mémoire, d'où une très grande rapidité, avec recopie périodique de la base dans un simple fichier texte ;
- support de « fermes de serveurs », par une distribution de la base : l'apparition d'un nouveau triplet est propagé vers les autres serveurs de la ferme.

Les macros que Sendmail doit envoyer au *milter* sont :

```
O Milter.macros.connect=j,{if_addr}
O Milter.macros.envfrom=i, {auth_authen}
O Milter.macros.helo={verify},{cert_subject}
```

Milter-greylist est configuré par un fichier `greylist.conf` :

```
peer 130.79.200.151
peer 2001:660:2402::175

acl whitelist addr 127.0.0.0/8
acl whitelist addr ::1/128

acl whitelist addr 130.79.0.0/16
acl whitelist addr 2001:660:2402::/48

lazyaw

dumpfreq 10m
timeout 5d
noaccessdb
greylist 1h
autowhite 5d
nospf

# This is a list of broken MTAs that break with greylisting. Copied from
# http://cvs.puremagic.com/viewcvs/greylisting/schema/whitelist_ip.txt?rev=1.12
acl whitelist addr 12.5.136.141/32    # Southwest Airlines (unique sender)
acl whitelist addr 207.171.168.0/24   # Amazon.com
```

Les principales directives sont :

[20] `http://hcpnet.free.fr/milter-greylist/`

`peer`	liste les adresses IPv4 ou IPv6 des autres serveurs de la ferme de serveurs SMTP. Les triplets sont échangés en temps réel (au fur et à mesure de leur entrée dans les listes grise et blanche) sur une connexion TCP avec le numéro de port 5252. Une autre directive (`syncaddr`) permet de fixer l'adresse IP et le port TCP d'émission, au cas notamment où ce serveur dispose de plusieurs adresses.
`acl`	ajoute une entrée fixe dans la liste grise ou la liste blanche, qui peut être une adresse IPv4 ou IPv6 (avec un masque pour préciser tout un réseau), un domaine, une adresse d'enveloppe pour un émetteur ou un destinataire. Des expressions régulières peuvent être utilisées. Il faut lister les adresses d'où on peut recevoir des messages légitimes (le réseau local), ainsi que les rares adresses des clients SMTP bogués. Le fichier exemple livré avec les sources répertorie déjà des adresses et fournit l'URL de référence[21] pour maintenir cette liste à jour.
`lazyaw`	normalement, milter-greylist teste les tentatives sur le triplet déjà mentionné (<adresse IP source, adresse d'enveloppe pour l'émetteur, adresse d'enveloppe pour le destinataire>). Cette option permet de ne faire les tests que sur l'adresse IP du client SMTP, ce qui élimine moins de *spam*, mais diminue la latence occasionnée car la mise en liste blanche est alors commune pour tous les émetteurs d'un domaine.
`dumpfreq`	péridiocité de la sauvegarde de la base (listes grise et blanche) dans le fichier texte, pour ne pas avoir à tout réapprendre à chaque démarrage.
`timeout`	durée de persistance dans la liste grise.
`noaccessdb`	laisser passer un message si Sendmail définit la macro `${greylist}`.
`greylist`	durée d'attente en liste grise imposée à la première tentative SMTP, avant de laisser passer le message.
`autowhite`	durée de persistance en liste blanche.
`nospf`	normalement lorsque milter-greylist est compilé avec le support SPF, il laisse passer immédiatement les clients SMTP ayant réussi le test SPF. Ce mot-clef désactive la fonctionnalité (voir 2.5.2).

Les valeurs par défaut donnent satisfaction et, dans la plupart des cas, l'administrateur se contentera d'ajouter les réseaux locaux dans la liste blanche.

L'installation de milter-greylist est un vrai soulagement pour l'administrateur système : c'est un programme sans histoire, et l'énorme impact de réduction du trafic dû au *spam* allège considérablement la charge des serveurs SMTP en frontal d'un domaine.

2.3 Filtres à heuristique

Le principe est de tester la présence dans le message de certaines caractéristiques typiques du *spam* comme l'utilisation exclusive de HTML[22]. Une pondération du test est effectué par un processus d'apprentissage, à partir d'un corpus de *spam* et de non-*spam* (les scores ne sont donc pas calculés par un humain, pour limiter la subjectivité).

Le plus connu de ces filtres est SpamAssassin[23].

Ces filtres détectent une grande proportion des *spams*. Ils n'ont pas besoin d'apprentissage ou de configuration. Mais, comme ils utilisent eux-mêmes de nombreux tests, il est préférable d'être conscient de la possibilité de changer les tests effectués et leurs scores.

Par exemple, SpamAssassin se configure via son fichier `local.cf`. On peut lui dire, par exemple :

```
# Ne pas changer le sujet des messages
rewrite_subject         0

# Temporaire, pour contourner une bogue dans le client Razor (Debian bug #176013)
score RAZOR_CHECK 0

# Ici, un test que nous considérons absurde : nous indiquons à
# SpamAssassin de ne pas l'effectuer (ce qui fera en outre gagner du temps).
# LACNIC being what it is, ipwhois.rfc-ignorant.org blacklists everything
# south of the RioGrande
# Debian bug #177570
score RCVD_IN_RFCI 0
```

[22] Une liste de tests se trouve en `http://spamassassin.apache.org/tests.html`.
[23] `http://www.spamassassin.org/`

```
# Partenaires dont on veut recevoir le courrier, même s'ils écrivent
# en "style spam"
whitelist_from  *@exemple.fr
whitelist_from  *@example.org
```

Ces filtres sont en général forts consommateurs de ressource machine.

Il faut prendre soin de mettre à jour régulièrement la base de tests (un peu comme avec un anti-virus). Un SpamAssassin vieux de deux ans n'attrape guère de *spams*, ceux-ci ayant évolué.

Simples pour les utilisateurs, ils offrent un bon taux de filtrage pour peu de travail de l'administrateur.

2.3.1 SpamAssassin avec Postfix

Nous prendrons SpamAssassin comme filtre.

Le *policy filter* ne transmet pas le corps du message, donc on ne peut pas l'utiliser tel quel pour un filtre à heuristiques. On peut lancer SpamAssassin depuis procmail, au moment du dépôt[24], ou bien via le MTA, ici Postfix[25] ou encore via Amavis, déjà cité. Ici, on n'utilisera pas d'intermédiaire, ni procmail, ni Amavis.

D'abord, on modifie la ligne "smtp" de `master.cf` pour y ajouter un *content filter* :

```
smtp  inet  n - n -  -  smtpd -o content_filter=spamassassin
```

Et on ajoute le filtre à la fin du même fichier :

```
spamassassin  unix  -   n   n  - -
   pipe   user=nobody \
   argv=/path/to/spamc -e \
      /path/to/postfix/sendmail -oi -f \
          ${sender} ${recipient}
```

2.3.2 SpamAssassin avec Sendmail

Il existe de nombreuses façons d'utiliser SpamAssassin avec Sendmail. Nous prendrons comme exemple le *milter* « spamass-milter[26] ». Attention : même si nous utilisons un *milter*, il faut tout de même installer SpamAssassin et tous ses modules Perl.

Les macros que Sendmail doit envoyer au *milter* sont :

```
O Milter.macros.connect=b,j,_,{daemon_name},{if_name},{if_addr}
```

En dehors de ces macros, et du paramétrage nécessaire de SpamAssassin lui-même, aucune configuration spécifique n'est nécessaire pour ce *milter*.

2.4 Filtres bayésiens

Cette technique fameuse a été introduite par [12].

Le principe est de de nourrir le moteur bayésien avec un corpus de *spam* et un corpus de non-*spam*. Il apprendra tout seul le vocabulaire des *spammeurs* et il utilisera ensuite les probabilités bayésiennes pour calculer si le message est un *spam* (de façon à éviter qu'un seul "V !agra" comme celui que je viens d'écrire fasse classer le message comme *spam*).

Dans le cas d'un filtre collectif, l'apprentissage est typiquement effectué par l'administrateur système.

Ces filtres sont très efficaces en utilisation individuelle et sont une des rares solutions qui, utilisée seule, peut filtrer la quasi-totalité des *spams*, après un bon apprentissage.

Étant fondés sur l'idée de vocabulaire *spam* ou *ham*, ils peuvent poser des problèmes en cas d'utilisation collective.

[24] `http://wiki.apache.org/spamassassin/UsedViaProcmail`
[25] `http://wiki.apache.org/spamassassin/IntegratedSpamdInPostfix`
[26] `http://savannah.gnu.org/projects/spamass-milt/`

Dans un environnement de petite taille et très homogène (par exemple une entreprise ou bien un département d'une Université où tout le monde travaille dans le même domaine), cela peut être acceptable. Mais cela ne l'est sans doute pas pour un gros FAI[27]. Pour celui-ci des bases individuelles sont impératives.

Notre recommandation est, dans le cas de tels filtres, de fournir aux utilisateurs la possibilité d'influencer le vocabulaire reconnu (la méthode la plus classique étant l'envoi d'un message à une adresse `this-is-a-spam@fai.fr`).

Nous rappelons que les filtres bayésiens sont surtout intéressants en utilisation individuelle. En collectif, surtout si la population concernée utilise des vocabulaires très différentes (cas d'une grande université multi-disciplinaire, par exemple), ces outils sont moins efficaces.

Ceci étant posé, nous allons voir comment intégrer bogofilter[28], l'outil le plus répandu pour réaliser le marquage bayésien, avec le MTA.

D'abord, il faut entrainer bogofilter en lui donnant un corpus de *spam* et de *ham*. Supposons que ces deux corpus sont sous forme de fichiers séparés dans les répertoires `/path/to/spam` et `/path/to/ham` :

```
 for i in /path/to/ham/*. ; do
      bogofilter -d /opt/var/spool/bogofilter/ -t -v -n < "$i"
done
 for i in /path/to/spam/*. ; do
      bogofilter -d /opt/var/spool/bogofilter/ -t -v -s < "$i"
done
```

-n indique que le message doit être considéré comme *ham*, -s qu'il doit être considéré comme *spam*.

Si les messages sont dans des *mailbox* Unix traditionnelles, `formail -s bogofilter [-s|-n]` convient. bogofilter a également une option -M qui peut servir.

Une fois le filtre bayésien entraîné[29], on peut l'intégrer dans le MTA.

2.4.1 Filtres bayésiens avec Postfix

L'utilisation de bogofilter dans Postfix est décrite dans une très bonne documentation qui figure dans la distribution de bogofilter, en `integrating-with-postfix`. Il faut écrire un petit script analogue à :

```
#!/bin/sh

FILTER=/usr/local/bin/bogofilter
FILTER_DIR=/var/spool/bogofilter
POSTFIX=/usr/sbin/sendmail
export BOGOFILTER_DIR=$FILTER_DIR

# Exit codes from <sysexits.h>
EX_TEMPFAIL=75
EX_UNAVAILABLE=69

cd $FILTER_DIR || \
{ echo $FILTER_DIR does not exist; exit $EX_TEMPFAIL; }

# Clean up when done or when aborting.
trap "rm -f msg.$$ ; exit $EX_TEMPFAIL" 0 1 2 3 15

# bogofilter -e returns: 0 for OK, nonzero for error
rm -f msg.$$ || exit $EX_TEMPFAIL
$FILTER -p -l -u -e > msg.$$ || exit $EX_TEMPFAIL

exec <msg.$$ || exit $EX_TEMPFAIL
rm -f msg.$$ # safe, we hold the file descriptor
exec $POSTFIX "$@"
```

Ensuite, on modifie le `master.cf` pour l'utiliser :

[27] D'une manière générale, les techniques anti-*spam* ne peuvent pas être les mêmes pour un serveur d'organisation, où la priorité est le travail des employés, et pour un FAI où la priorité est de fournir aux clients le service auquel ils se sont abonnés. En général, un FAI a moins de latitude pour filtrer sévèrement, de crainte des faux positifs.

[28] `http://www.bogofilter.org/`

[29] Vous pouvez regarder le contenu de la base avec `bogofilter -vvv < UNMESSAGE` pour voir les notes obtenues par les mots contenus dans `LEMESSAGE` ou bien utiliser `bogoutil -d` pour afficher toute la base ou encore `bogoutil -w < UNMESSAGE` pour afficher les occurrences de chaque mot dans le corpus de *spam* et de *ham*.

```
smtp  inet  n  - n       -       - \
     smtpd -o content_filter=bogofilter:
...
bogofilter    unix  -  n  n  -   -  \
      pipe   flags=R user=bogo \
      argv=/local/sbin/postfix-bogofilter.sh \
             -f ${sender} -- ${recipient}
```

bogofilter ne fait rien d'autre au message que d'ajouter un en-tête :

```
X-Bogosity: Ham, tests=bogofilter, spamicity=0.379443, version=0.96.0
```

Et il reste à filtrer sur cet en-tête, dans le MUA ou dans un fichier procmail collectif.

Si on utilise un filtre bayésien pour un large public, il faut absolument permettre à ce public de l'influencer. Le plus simple est de créer deux adresses `ham@monsite.fr` et `spam@monsite.fr` et de demander aux utilisateurs de renvoyer[30] le courrier à ces deux adresses pour mettre à jour la base de bogofilter. Le fichier `aliases` contiendra :

```
spam: "| sudo -u bogo bogofilter -s"
ham: "| sudo -u bogo bogofilter -n"
```

2.4.2 Filtres bayésiens avec Sendmail

L'intégration du marquage bayésien est réalisée grâce à un *milter* dont la seule fonction est d'appeler bogofilter. Il s'agit de « milter-bogom[31] ».

Ce milter est malheureusement moins abouti que d'autres ; sa mise en place recèle quelques pièges. Notamment, milter-bogom utilise l'identité d'un utilisateur non privilégié, mais il ne sait pas utiliser le répertoire de cet utilisateur par défaut. La solution est alors :
– de modifier le script d'installation pour forcer l'utilisateur par défaut avant de lancer le *milter* ;
– de modifier le script d'installation pour forcer la variable HOME ou BOGOFILTER_DIR avant de lancer le *milter* ;
– ou alors de déplacer la base dans le répertoire de l'utilisateur *root*.
Les macros que Sendmail doit envoyer au *milter* sont :

```
O Milter.macros.connect=b,j,_,{daemon_name},{if_name},{if_addr}
```

Milter-bogom est configuré par un fichier `bogom.conf` :

```
policy pass
# subject_tag "[SPAM] "
training 0
# body_limit 1m
```

Les principales directives sont :

`policy`	spécifie le traitement des messages identifiés comme *spam*. Compte-tenu des faux positifs possibles, il est prudent de ne pas rejeter ou supprimer les messages, mais de les laisser passer.
`subject_tag`	modifie le sujet en cas de reconnaissance d'un *spam*. Normalement, un champ X-Bogosity est ajouté à l'en-tête et ceci devrait suffire.
`training`	indique à bogofilter si la base doit être laissée telle qu'elle (0), ou être modifiée suite à la reconnaissance d'un *spam* ou d'un *ham* (1). Attention : si vous indiquez 1, la base peut croître énormément et elle nécessitera alors une maintenance plus importante.
`body_limit`	pour des raisons d'efficacité, il peut être intéressant de ne pas passer à bogofilter des messages de taille trop importante. Par défaut, il n'y a pas de limite, tous les messages sont marqués.

Milter-bogom appelle bogofilter, et ajoute ensuite un champ dans l'en-tête du message. Ce champ peut contenir les trois valeurs fixes :

– `X-Bogosity: Yes, tests=bogofilter`
– `X-Bogosity: No, tests=bogofilter`

[30] *bounce* ou *redirect*, pas *forwarding* afin de ne pas ajouter leurs propres en-têtes. Cela peut s'automatiser dans des logiciels comme Thunderbird.
[31] `http://www.usebox.net/jjm/bogom/`

– `X-Bogosity: Unsure, tests=bogofilter`

Ces valeurs sont moins précises que ce que bogofilter sait indiquer (bogofilter peut notamment indiquer la probabilité). Ceci est dû au mode de fonctionnement du *milter*, qui teste uniquement le code de retour de bogofilter et ajoute lui même le champ d'en-tête.

Pour effectuer des traitements complexes, comme c'est souvent le cas dans la lutte anti-*spam* (ou anti-virus), il vaut mieux que Postfix passe le message à un programme spécialisé. Certains programmes comme SpamAssassin peuvent être utilisés comme ceci mais on préfère souvent passer par un intermédiaire, chargé de recevoir le courrier de Postfix, de le soumettre à un certain nombre de programme de tests, puis de le réinjecter dans Postfix. Les programmes de tests n'ont donc pas à connaitre SMTP et l'intermédiaire n'a pas à connaitre toutes les technologies anti-*spam* et (anti-virus). Le plus connu de ces intermédiaires est amavis-new[32].

Voici comment configurer Postfix pour passer tout le courrier à Amavis (il faudra ensuite configurer Amavis pour chaque test). Dans le `master.cf`, on ajoute les instructions nécessaires au lancement d'Amavis, et d'un Postfix supplémentaire qui récupérera les messages après filtrage :

```
# Amavis new
smtp-amavis unix -      -       n       -       2  smtp
    -o smtp_data_done_timeout=1200
    -o smtp_send_xforward_command=yes
    -o disable_dns_lookups=yes
    -o max_use=20
127.0.0.1:10025 inet n  -       n       -       -  smtpd
    -o content_filter=
    -o local_recipient_maps=
    -o relay_recipient_maps=
    -o smtpd_restriction_classes=
    -o smtpd_delay_reject=no
    -o smtpd_client_restrictions=permit_mynetworks,reject
    -o smtpd_helo_restrictions=
    -o smtpd_sender_restrictions=
    -o smtpd_recipient_restrictions=permit_mynetworks,reject
    -o mynetworks_style=host
    -o mynetworks=127.0.0.0/8
    -o strict_rfc821_envelopes=yes
    -o smtpd_error_sleep_time=0
    -o smtpd_soft_error_limit=1001
    -o smtpd_hard_error_limit=1000
    -o smtpd_client_connection_count_limit=0
    -o smtpd_client_connection_rate_limit=0
    -o receive_override_options=no_header_body_checks,no_unknown_recipient_checks
```

Ensuite, on dira à Postfix de soumettre les messages à Amavis, en ajoutant dans le `main.cf` :

```
content_filter=smtp-amavis:[127.0.0.1]:10024
```

2.5 SPF

SPF [13] fait partie d'une famille de protocoles (qui comprend aussi Sender-ID et DKIM[33]) qui visent à permettre l'**authentification** du courrier électronique. L'authentification est utile à la lutte contre le *spam* mais elle n'en fait pas directement partie.

Il est en effet fondamental de se rappeler que ces techniques sont des solutions d'**authentification**, pas de lutte contre le *spam* à proprement parler.

Une analogie permettra de comprendre : les documents d'identité ne sont pas un "marqueur de confiance". Les délinquants peuvent avoir une carte d'identité eux aussi. Mais l'obligation de transparence profitera plus aux émetteurs légitimes qu'aux *spammeurs*.

SPF[34] et Sender-ID permettent de tester si un serveur de messagerie est autorisé à émettre pour le compte d'un domaine donné. Cela se fait par la publication dans le DNS d'un enregistrement qui liste les serveurs de messagerie autorisés pour un domaine.

[32] `http://www.ijs.si/software/amavisd/`
[33] *Domain Keys Identified Mail*
[34] SPF, *Sender Policy Framework* est présenté en [13].

Par exemple, pour le domaine `freebsd.org`, l'enregistrement SPF[35] est `v=spf1 ip4 :216.136.204.119 ~all`, ce qui signifie que le courrier venant de 216.136.204.119 est authentique et celui qui vient d'ailleurs ne l'est probablement pas (~ signifie « sans doute pas »).

Les deux techniques se différencient surtout par le choix de l'identité testée. SPF teste le MAIL FROM de l'enveloppe (RFC 2821) et Sender-ID les en-têtes (RFC 2822).

Les administrateurs des serveurs peuvent agir sur deux points : publier des enregistrements SPF dans le DNS et les tester en entrée. C'est ce deuxième point que nous couvrons ici.

L'authentification du courrier électronique par la vérification des adresses IP du serveur émetteur devrait contribuer dans le futur à limiter et à mieux contrôler le *spam*. Ceci suppose probablement la création de services au dessus de l'authentification, par exemple des listes blanches privées, des services de réputation, des services d'accréditation, etc.

À l'heure actuelle, le bénéfice pratique immédiat est très faible, peu de domaines publiant des enregistrements SPF.

L'attention des administrateurs est attiré sur le fait que Sender-ID[36] est très récent et peu testé et que les deux techniques ne s'appuient pas encore sur une norme stable[37].

En outre, Sender-ID semble soumis à un brevet, avec une licence d'utilisation incompatible avec certaines licences de logiciel libre.

Notre recommandation est de tester SPF en entrée, sans lui accorder un poids trop fort, vue la maturité de cette technique.

L'authentification est une étape importante dans toute sécurisation du courrier. SPF est de loin la technique la mieux comprise et la plus déployée.

D'autre part, bien que cela concerne un autre document, nous recommandons de publier des enregistrements SPF[38].

2.5.1 Postfix

La meilleure solution est d'utiliser le *policy filter* [39] qui est apparu avec Postfix 2.1.

```
my $query = eval { new Mail::SPF::Query (ip    =>$attr{client_address},
                                         sender=>$attr{sender},
                                         helo  =>$attr{helo_name}) };
if ($@) {
  syslog(info=>"%s: Mail::SPF::Query->new(%s, %s, %s) failed: %s",
         $attr{queue_id}, $attr{client_address}, $attr{sender}, $attr{helo_name}, $@);
  return "DUNNO";
}
my ($result, $smtp_comment, $header_comment) = $query->result();

syslog(info=>"SPF %s: smtp_comment=%s, header_comment=%s",
       $result, $smtp_comment, $header_comment);
my $iftest = "";
if ($test) {
    $iftest = " (running in test mode so always accepting)";
}
print STDOUT "action=PREPEND Received-SPF: \
      $hostname: $result$iftest: smtp_comment=$smtp_comment, header_comment=$header_comment\n";

if ($test) { return "DUNNO"; }

if    ($result eq "pass")  { return "DUNNO"; }
elsif ($result eq "fail")  {
         return "REJECT " . ($smtp_comment || $header_comment);   }
elsif ($result eq "error") { return "450 temporary failure: $smtp_comment"; }
else                       { return "DUNNO"; }
# unknown, softfail, neutral and none all return DUNNO
```

[35] Qu'on peut voir avec `dig TXT freebsd.org`..

[36] `http://www.microsoft.com/mscorp/safety/technologies/senderid/default.mspx`

[37] Le projet de norme actuel pour SPF est `http://www.ietf.org/internet-drafts/draft-schlitt-spf-classic-02.txt`, qui a été accepté par l'IESG mais pas encore publié comme RFC.

[38] Pour limiter les risques, il est recommandé de les terminer par `?all` ou `ãll` et/ou d'utiliser les techniques écrites en `http://spf.idimo.com/how_to-s/how_to_track.html`.

[39] `http://www.postfix.org/SMTPD_POLICY_README.html`

On dit à Postfix, dans `master.cf`, de lancer le *policy filter* SPF[40] :

```
policy  unix  - n  n  -  -  spawn  \
       user=nobody arg \
       v=/usr/bin/perl \
           /usr/local/sbin/postfix-policyd-spf.pl -t
```

On configure Postfix, dans `main.cf`, pour soumettre les sessions SMTP au test :

```
smtpd_recipient_restrictions = permit_mynetworks, ...
       check_policy_service unix:private/policy, ...
```

Le script `postfix-policyd-spf.pl` va alors décider de rejeter le courrier qui n'est pas authentique, ou seulement de le marquer pour filtrage ultérieur.

2.5.2 Sendmail

Le support de SPF avec Sendmail peut prendre plusieurs formes :

1. mettre automatiquement les tentatives authentifiées par SPF dans la liste blanche du *greylisting* ;
2. marquer et éventuellement rejeter les messages dont l'authentification SPF échoue ;
3. ajouter l'authentification SPF dans les critères de pondération de reconnaissance de *spam*, et en particulier les filtrages heuristiques de type SpamAssasin.

La première solution est facile à réaliser avec milter-greylist (cf 2.2.2). Si ce *milter* est compilé avec le support de SPF (disponible depuis la version 2.0), il laisse rentrer tous les messages dûment authentifiés. Cette solution n'est toutefois pas conseillée, car beaucoup de *spammeurs* ont compris la technique et on se retrouve alors avec le paradoxe qu'avec SPF, les *spams* rentrent plus vite...

Le marquage et le rejet des messages peut être réalisé via le *milter* « sid-milter[41] » émanant des auteurs de Sendmail. À l'heure actuelle, ce *milter*, qui est encore en version 0.2, est contrôlé par la ligne de commande et permet de rejeter les messages suivant plusieurs politiques :

– par défaut, aucun rejet n'est effectué, seul le marquage est effectué ;
– rejet si les authentifications Sender-Id et SPF échouent toutes les deux ;
– rejet si l'une des authentifications Sender-Id ou SPF échoue ;
– rejet sauf si l'une des deux authentifications passe ;
– rejet sauf si les deux authentifications passent.

La dernière solution, que nous recommandons par rapport aux deux premières, consiste à laisser SpamAssassin (voir 2.3.2) réaliser le test SPF lui-même, et en tenir compte dans son évaluation globale.

2.6 Filtre comportemental

Ce type de filtres regarde le comportement du serveur distant (nombre de messages envoyés par unité de temps, par exemple). Le *rate-limiting* en est un exemple.

L'idée est que les messages ordinaires ne sont émis que un par un et les *spams* par paquets.

Ce genre de filtres est très délicat car rien ne distingue un *spammeur* d'un serveur de listes de diffusion.

Notre recommandation est que, s'il y a un tel filtre, il doit donc être utilisé en limitation de débit, pas en bloquage.

2.7 Existence du domaine émetteur

De nombreux *spammeurs* envoient encore du courrier avec une adresse d'émetteur non-existante[42].

Idéalement, on contrôle également la validité du MX dudit domaine (contrôler qu'il n'est pas dans 127.0.0.0/8, par exemple, comme on le voit dans l'exemple en 2.1.1).

[40] On peut télécharger celui-ci en `http://spf.pobox.com/postfix-policyd.txt`.
[41] `http://sendmail.net/sid-milter/`. Ce *milter* traite également l'authentification Sender-ID.
[42] Comme dans n'importe quelle activité délinquante, il existe des professionnels et des amateurs.

Cela rejette une certaine quantité de *spam*.

Cela implique un peu de trafic DNS (mais qui aurait probablement eu lieu de toute façon, au moment de la réponse).

Notre recommandation est d'utiliser cette technique.

La gêne est très limitée car, si le domaine émetteur n'existe pas, il sera difficile (mais pas impossible, grâce au `Reply-To:`) de répondre au courrier, de toute façon.

2.7.1 Postfix

On peut utiliser la directive `reject_unknown_sender_domain` de Postfix

2.7.2 Sendmail

Les règles anti-*spam* du Kit Jussieu implémentent ce test automatiquement.

2.8 L'enfer

Par souci de complétude, voici quelques techniques que nous avons choisi de ne pas mettre en avant, parce qu'elles ont trop d'inconvénients ou parce qu'elle sont trop récentes. A utiliser avec précautions.

2.8.1 Listes noires

Le principe est de lister, dans un serveur accessible (en général par le protocole DNS), des machines considérées comme émetteurs de *spam*. Certaines listes contiennent des relais ouverts, qui n'émettent pas que du *spam*.

Il existe des listes noires publiques et des privées, typiquement accessibles sur abonnement.

Avec les MTA actuels, ce test peut être effectué dans la session SMTP et donc mener à un rejet avant même la transmission.

Mais les listes noires sont de qualité très variable. Il existe beaucoup de cas de listes mal gérées, abandonnées, ou bien agissant de manière peu sérieuse (ajout rapide mais retrait quasiment impossible, etc) [14].

Notre recommandation est de bien choisir la liste utilisée. Il est fortement recommandé d'utiliser des listes ayant elles-mêmes des bonnes pratiques :

– administrateur qui répond et qui réagit,
– critères d'ajout et de retrait publics et non-discriminatoires,
– critères raisonnables.

Dans le doute, il est plus prudent de ne pas utiliser une liste publique, étant donné la faible fiabilité de beaucoup. Quant on utilise une liste extérieure, il est nécessaire de suivre son activité : la qualité évolue dans le temps.

Si on gère sa propre liste noire (un gros travail), les critères de qualité ci-dessus s'appliquent bien évidemment aussi.

S'il est difficile de faire une synthèse simple des listes noires, vue leur variété, nous préférons recommander la prudence, vus les accidents qui ont déjà eu lieu.

Une note en passant : si **vous** vous retrouvez sur une liste noire, vous risquez de ne plus pouvoir envoyer de courrier à certains correspondants. Les listes noires publiques sont par définition accessibles de l'extérieur et peuvent être consultées avec les outils DNS classiques, comme dig. Sinon, elles fournissent en général une interface Web pour interroger leur contenu (comme `http://www.nl.sorbs.net/lookup.shtml` pour Sorbs). L'outil rblcheck permet également de tester un grand nombre de listes noires d'un coup (ici avec l'adresse IP d'un *spammeur* français connu) :

```
% rblcheck 62.23.133.234
checking 62.23.133.234
62.23.133.234 not RBL filtered by dev.null.dk
62.23.133.234 not RBL filtered by blackholes.mail-abuse.org
62.23.133.234 not RBL filtered by relays.mail-abuse.org
62.23.133.234 not RBL filtered by dialups.mail-abuse.org
62.23.133.234 not RBL filtered by relays.ordb.org
62.23.133.234 not RBL filtered by multihop.dsbl.org
62.23.133.234 not RBL filtered by xbl.spamhaus.org
```

```
62.23.133.234 not RBL filtered by sbl.spamhaus.org
62.23.133.234 RBL filtered by dnsbl.njabl.org
62.23.133.234 not RBL filtered by dul.dnsbl.sorbs.net
62.23.133.234 not RBL filtered by l1.spews.dnsbl.sorbs.net
```

2.8.2 PTR

Un enregistrement PTR du DNS permet la traduction de l'adresse IP du serveur émetteur en nom, sans forcément vérifier sa cohérence avec le domaine émetteur.

L'ajout de tels enregistrements n'est pas entièrement sous le contrôle du domaine émetteur (si son prestataire ne lui délègue pas `in-addr.arpa`, par exemple) et celui-ci, même légitime, peut être dans l'incapacité de remplir cette obligation.

Notre recommandation est d'éviter ce test.

2.8.3 Filtre à mots-clés

Il s'agit de filtres qui décident de manière binaire sur la présence d'un mot-clé ("Viagra" $\Longrightarrow$ *spam*).

Ils ne sont mentionnés que pour mémoire.

Ils doivent être évités absolument car les risques de faux positifs sont énormes.

2.8.4 HELO / CSV

Un MTA SMTP annonce son nom avec la commande HELO (ou EHLO s'il est en Extended SMTP).

Beaucoup de spams sont émis avec une commande HELO/EHLO comportant l'adresse IP du serveur destinataire. Cela permet de maquiller en partie le dernier entête Received :. Il est intéressant de configurer un serveur SMTP pour refuser au moins les HELO qui comportent sa propre adresse IP, car le risque de faux positif est pratiquement inexistant et le nombre de spams filtrés par ce moyen est loin d'être négligeable.

De même on peut choisir de refuser les HELO comportant le nom de domaine du serveur SMTP. Cependant, il devient alors important de bien choisir les priorités des règles sur le serveur afin d'accepter le relayage local de MUA SMTP, qui pratiquent souvent un HELO avec la partie droite de l'adresse RFC 822.

Les tests CSV ([43]) ajoutent un test de vraisemblance sur le nom : correspond t-il vraiment au domaine ? Contrairement à SPF ou à DKIM, CSV n'authentifie pas le domaine émetteur du message mais le domaine du serveur de messagerie (qui peuvent être différents, par exemple dans le cas d'un prestataire hébergeant de nombreux clients).

Des directives de configuration permettent de tester le nom annoncé par le serveur : avec les seuls tests HELO classique, sans CSV, le nombre de faux positifs est très important. Peu de sites savent modifier le HELO pour le rendre correct. Cela changera probablement dans le futur.

Notre recommandation est de ne pas les utiliser.

2.8.5 *token*/mot de passe

Il s'agit d'inclure dans l'adresse du destinataire un mot de passe ou bien d'utiliser un système de défi/réponse (par exemple par un test de Turing).

Le *spammeur* (ou plutôt son logiciel) ne pourra pas connaitre ce mot de passe, ni passer ce test.

Ces techniques n'ont pas de faux négatif (à moins que des *spammeurs* un jour n'embauchent des milliers de personne dans un pays à faible coût de main d'œuvre).

Un certain nombre d'utilisateurs légitimes refusera de passer le test ou n'y arrivera pas. Il y aura donc beaucoup de faux positifs.

Ces techniques n'ont d'intérêt que pour des destinataires très populaires et qui reçoivent de toute façon trop de courrier, même légitime. Il faut en effet être certain que l'émetteur acceptera l'épreuve imposée. Une liste de diffusion d'entraide peut par

[43] *Certified Server Validation,* `http://mipassoc.org/csv/index.html`

exemple mettre en œuvre une telle technique (avec liste blanche pour les adresses de ses abonnés) en comptant que les auteurs de message ont une puissante motivation pour écrire à la liste.

Fondamentalement, ces techniques tendent à reporter le coût du traitement du *spam* sur l'émetteur légitime et innocent (alors qu'il est actuellement supporté par le destinataire). Il faut les voir comme la mise en œuvre d'une politique "liste blanche seulement" et pas comme une technique de filtrage.

2.8.6 Filtre à résumés ou à empreintes

Les filtres à empreinte, comme Razor, calculent une empreinte du message qui leur est soumis et signalent s'il a déjà été dénoncé comme *spam.*

Ils donnent peu de faux positifs.

En revanche, les faux négatifs sont nombreux, car beaucoup de *spams* ne sont pas encore signalés lorsque votre serveur interroge Razor (et le message varie parfois suffisamment pour que l'empreinte soit différente).

Une solution à ce problème est de retarder le courrier (comme le fait le *greylisting*).

3 La combinaison des tests

L'attention des administrateurs est attirée sur le fait que les tests ne sont pas forcément à utiliser en « tout ou rien ». Compte-tenu de la pertinence (pas toujours forte) des tests et compte-tenu du risque de faux positifs, il est au contraire courant de combiner les tests.

– « tout ou rien » en refus : c'est le comportement classique des serveurs qui utilisent une liste noire (mais ce n'est pas obligatoire). Si un test rate, le message est refusé. Il faut donc être sûr de ses tests.
– privilège : c'est le comportement classique des serveurs qui utilisent une liste blanche. Si le test réussit, le message passe. Il n'y a pas de risque de faux positifs mais on peut avoir des faux négatifs. Par exemple, une liste blanche de domaines n'a guère d'intérêt si on ne teste pas l'authenticité du domaine émetteur (avec SPF ou DKIM). Beaucoup de *spams* ou de vers prétendent venir d'`amazon.com` dans l'espoir de passer par la voie privilégiée.
– *scoring* : c'est le comportement de logiciels comme SpamAssassin (procmail peut le faire aussi) pour combiner leurs différents tests. Évitant les inconvénients du "tout ou rien", le *scoring* est très recommandé. Mais il est coûteux en ressources machine.

La méthode classique est de procéder à quelques tests "tout ou rien", puis de soumettre au *scoring* les messages qui sont passés.

La bonne pratique recommandée est de combiner plusieurs techniques mais pas trop, la complexité étant source d'imprévus. Regardez ce que vous pouvez maitriser.

4 Résultats des tests

Un fois le message classé comme *spam*, grâce à la combinaison des tests, il reste à décider de la manière dont il est traité.

Cette manière dépend évidemment des tests effectués. Certains tests sont plus fiables que d'autres et peuvent donc justifier des mesures plus drastiques.

D'autre part, l'action suite à un test raté peut être d'effectuer d'autres tests, plus coûteux, et qu'on ne souhaite pas faire pour tout message.

4.1 Rejet dans la session SMTP

– cela économise la bande passante car on ne recevra pas le message,
– cela reporte la responsabilité d'un éventuel DSN (Delivery Status Notification, le message généré suite à un rejet, voir RFC 3461) sur l'émetteur[44].
– mais cela ne permet pas de garder une copie du message (donc de rattraper un faux positif, ou simplement d'enquêter sur un rejet),

[44] Après la session SMTP, le choix d'émettre un DSN ou pas est très délicat car l'adresse de l'émetteur étant souvent mensongère, on risque d'embêter un innocent.

– avec la plupart des serveurs SMTP, les possibilités de tests à effectuer pendant la session SMTP sont très limitées (mais cela change avec des interfaces comme les *milters* de Sendmail, le "policy server" de Postfix, ou bien le futur OPES[45], qui permettront de brancher n'importe quel programme sur la session SMTP).

Ce rejet est donc à réserver aux cas où on est raisonnablement certain de sa pertinence.

4.2 Rejet silencieux

Cela brise l'attente des utilisateurs qui comptent que le message sera remis à son destinataire ou bien qu'ils en seront avertis.

Cette alternative "dépose ou préviens" était un principe cardinal de la messagerie, mais qui devra probablement être abandonné, en raison du nombre de *joe jobs* [46].

Idéalement, une trace des messages ainsi détruits devraient être gardés, pour pouvoir mettre en oeuvre des techniques comme le *Message Tracking* (RFC 3885).

Il faut noter que toutes les "pertes de message" telles que dénoncées par les utilisateurs ne sont pas forcément des rejets silencieux par un fournisseur. La cause de la perte peut être ailleurs (effacement trop rapide d'un message par le récepteur humain ou d'un DSN par l'émetteur humain).

4.3 Rejet par envoi d'un DSN (*bounce* ou rebond)

C'est la méthode traditionnelle de la messagerie Internet. Mais, en présence des *joe jobs*, elle risque fort de frapper un innocent (comme on le voit avec les anti-virus qui émettent à tort des DSN). Si un message a été classé comme *spam*, son adresse d'émetteur est probablement fausse et il ne sert donc à rien d'envoyer un DSN.

4.4 Mise dans une boîte *spam*

On ne détruit pas le message, mais on donne à l'utilisateur une chance de rattraper les faux positifs. Peu de gens consultent régulièrement leur boîte à *spam* (le but du filtrage automatique étant justement de ne pas perdre de temps avec la lecture des *spams*) mais elle fournit un filet de sécurité

Comme peu d'utilisateurs la lisent, il est prudent de la vider de temps en temps automatiquement.

4.5 Simple marquage

Le serveur ne prend aucune décision, il met simplement une note dans le message. Cette technique donne un complet contrôle à l'utilisateur, mais elle va le forcer à télécharger des *spams*.

On note qu'un prestataire de messagerie peut laisser le choix entre le simple marquage et la mise dans une boîte *spam* à l'utilisateur. C'est relativement simple à gérer.

Pour marquer, modifier le corps du message ou bien le champ Subject : est dangereux techniquement (cela casse les signatures) et juridiquement. Il est préférable d'ajouter un en-tête (comme "X-Provider-spamscanner : YES"), que tous les logiciels de messagerie savent utiliser pour filtrer. Si votre logiciel de messagerie ne sait pas configurer ses filtres sur des en-têtes choisis par l'utilisateur (exemple : vieux Netscape ou Eudora), il est sans doute temps d'en changer.

5 Conclusion

Pour les prochaines années, il est hélas certain que le *spam* ne va pas disparaitre et l'administrateur système doit s'habituer à une lutte permanente. Toute configuration de messagerie nouvelle, aujourd'hui, doit intégrer le problème du *spam* dès le début.

Dans tous les cas, lorsqu'on déploie une nouvelle technique, par exemple un nouveau test, il est primordial d'avoir une période d'essai, où on se contente de marquer sans détruire, afin d'évaluer le risque de faux positifs.

[45] RFC 3835 ou `http://www.ietf.org/html.charters/opes-charter.html`

[46] Message prétendant venir d'une adresse, alors que son origine est sans lien avec le prétendu expéditeur. La plupart des *spams* sont des *joe jobs*.

A Annexe

A.1 Anti-virus

Il serait dommage d'aborder les outils anti-*spam* sans mentionner l'excellent anti-virus libre ClamAV[47].

Cet anti-virus (pas pour les serveurs Unix, bien sûr, mais pour les machines MS-Windows qu'ils servent) est très souple d'utilisation, performant et efficace. La mise à jour de la base anti-virale est très simple (comme les virus évoluent vite, il faut s'assurer que la base de Clamav est remise à jour très régulièrement avec la commande freshclam). Cependant, la difficulté majeure reste toutefois de maintenir le moteur à jour. Il faut surveiller les logs pour savoir quand une nouvelle version est nécessaire, et réaliser la mise à jour le plus rapidement possible. Là encore, une distribution d'Unix réactive et qui vous fournit des mises à jour prêtes à l'emploi est un plus significatif dans l'administration quotidienne de cet outil.

A.1.1 Postfix

Le plus simple est de lancer Clamav par Amavis, le code nécessaire figure déjà en commentaires dans la configuration d'Amavis, il suffit de le décommenter dans `amavisd.conf` :

```
# ### http://www.clamav.net/
['ClamAV-clamd',
   \&ask_daemon, ["CONTSCAN {}\n", "/var/run/clamav/clamd"],
   qr/\bOK$/, qr/\bFOUND$/,
   qr/^.*?: (?!Infected Archive)(.*) FOUND$/ ],
# NOTE: the easiest is to run clamd under the same user as amavisd; match the
# socket name (LocalSocket) in clamav.conf to the socket name in this entry
# When running chrooted one may prefer: ["CONTSCAN {}\n","$MYHOME/clamd"],
```

Postfix va alors remettre tous les courriers à Amavis qui effectuera un certain nombre de tests, dont le test Clamav. Amavis réinjectera ensuite le courrier à Postfix.

A.1.2 Sendmail

La distribution ClamAV est fournie avec un *milter* très abouti pour Sendmail, qui soumet le message au démon `clamd` pour tester la présence de virus.

Ce *milter* est très simple à mettre en place. Il utilise le fichier `clamd.conf` général. Aucune configuration spécifique n'est requise.

Références

[1] Stéphane Bortzmeyer, *Généralités sur le spam*. January 2004. `http://www.ddm.gouv.fr/IMG/pdf/afnic_180204.pdf`.

[2] Direction du Développement des Médias (Premier Ministre), *Groupe de contact des acteurs de la lutte contre le spam*. `http://www.ddm.gouv.fr/rubrique.php3?id_rubrique=63`.

[3] CERTA, *Limiter l'impact du spam*. `http://www.certa.ssi.gouv.fr/site/CERTA-2005-INF-004/index.html`.

[4] Stéphane Bortzmeyer. La publicité dans votre boîte aux lettres. `http://1997.jres.org/articles/res5/bortzmeyer/Spam/index.html`. JRES, 1997.

[5] CRU, *Ressources antispam*. `http://www.cru.fr/antispam/`.

[6] Brian McWilliams. *Spam Kings*. Numéro 0-596-00732-9 dans ISBN. O'Reilly, October 2004.

[7] The Honeynet Project & Research Alliance, *Know your Enemy : Tracking Botnets*. March 2005. `http://www.honeynet.org/papers/bots/`.

[8] Bruce Schneier. *Secrets and lies*. Numéro 0-471-25311-1 dans ISBN. John Wiley, 2000.

[9] Bruce Schneier. *Beyond fear*. Numéro 0-387-02620-7 dans ISBN. Copernicus, September 2003.

[47] `http://www.clamav.net`

[10] Pierre David et Jacky Thibault et Sébastien Vautherot, *Kit Jussieu*. http://www.kit-jussieu.org/.

[11] *Greylisting*. http://www.greylisting.org/.

[12] Paul Graham, *A Plan for Spam*. August 2002. http://www.paulgraham.com/spam.html.

[13] Meng Weng Wong, *SPF : Sender Policy Framework*. http://spf.pobox.com/.

[14] Frédéric Aoun et Bruno Rasle. Listes noires, listes blanches : efficacité et utilisation. http://www.halte-au-spam.com/ddm-blacklists.htm. DDM, November 2004.