

# Lutte anti-spam concrète et pratique avec du logiciel libre

Stéphane Bortzmeyer  
AFNIC  
bortzmeyer@nic.fr

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

Ce document est distribué sous les termes de la GNU Free Documentation License

<http://www.gnu.org/licenses/licenses.html#FDL>.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation ; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.

# Objectifs

Objectifs de ce tutoriel :

- initier aux techniques anti-spam
- fournir des réponses concrètes
- utiliser des logiciels libres
- contexte : un MTA « tête de réseau » d'une université ou d'un laboratoire

Application aux deux logiciels les plus utilisés dans notre communauté :

- Postfix
- Sendmail

# Définissons le problème

# Définissons le problème

- 1 Vous recevez trop de *spam*,

# Définissons le problème

- 1 Vous recevez trop de *spam*,
- 2 Le courrier est important (donc vous ne voulez pas tuer le malade),

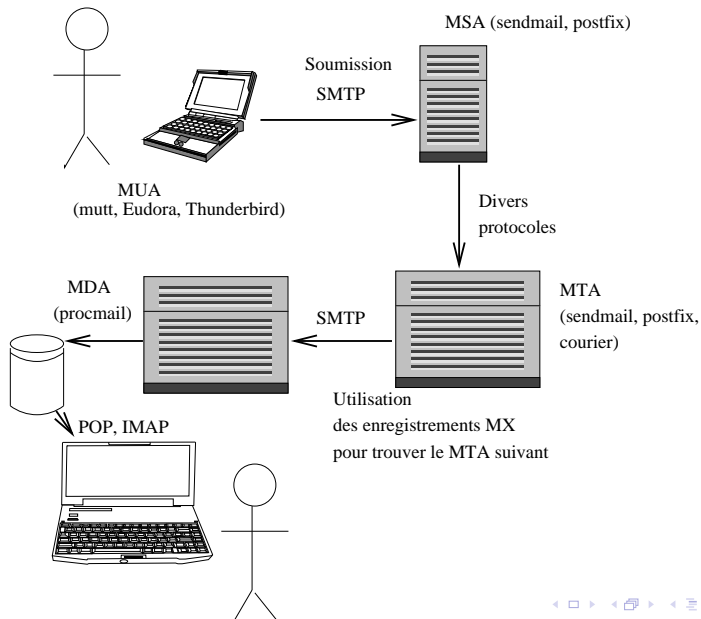
# Définissons le problème

- 1 Vous recevez trop de *spam*,
- 2 Le courrier est important (donc vous ne voulez pas tuer le malade),
- 3 Vous vous méfiez des charlatans (nombreux dans ce domaine),

# Définissons le problème

- 1 Vous recevez trop de *spam*,
- 2 Le courrier est important (donc vous ne voulez pas tuer le malade),
- 3 Vous vous méfiez des charlatans (nombreux dans ce domaine),
- 4 Vous avez postfix ou sendmail et vous voulez utiliser du logiciel libre.

# Petit rappel : transport du courrier



## Petit rappel : SMTP

Les messages du protocole SMTP spécifient l'**enveloppe**.

```
~ % telnet mailhost.u-strasbg.fr smtp
Trying 2001:660:2402::153...
Connected to mailhost.u-strasbg.fr.
Escape character is '^]'.
220 mailhost.u-strasbg.fr ESMTP Sendmail 8.13.3/jtpda-5.5pre1 ready at 1
HELO mailhost.nic.fr
250 mailhost.u-strasbg.fr Hello batilda.nic.fr [IPv6:2001:660:3003:8::4]
MAIL FROM:<bortzmeyer@nic.fr>
250 2.1.0 <bortzmeyer@nic.fr>... Sender ok
RCPT TO:<Pierre.David@crc.u-strasbg.fr>
250 2.1.5 <Pierre.David@crc.u-strasbg.fr>... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself
```

## Petit rappel : format du courrier

Le RFC 2822 décrit le **contenu** : en-tête et corps.

```
Received: from vagabond.noos.fr (rob67-1-82-231-68-196.fbx.proxad.net [
    by smtp1-g19.free.fr (Postfix) with ESMTTP id D8C2B4D655
    for <bortzmeyer@nic.fr>; Fri, 7 Oct 2005 17:19:32 +0200 (CEST)
Date: Fri, 7 Oct 2005 17:19:30 +0200
To: Stephane Bortzmeyer <bortzmeyer@nic.fr>
Message-ID: <20051007151930.GH42064@vagabond.ma.maison>
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>
```

Les en-têtes Received se lisent de bas en haut et ils peuvent être mensongers.

# Fonctionnement du *spam*

- 1 Les *spammeurs* sous-traitent, comme dans le BTP,

## Fonctionnement du *spam*

- ① Les *spammeurs* sous-traitent, comme dans le BTP,
- ② Ils utilisent un réseau de zombies. *A botnet is comparable to compulsory military service for Windows boxes*

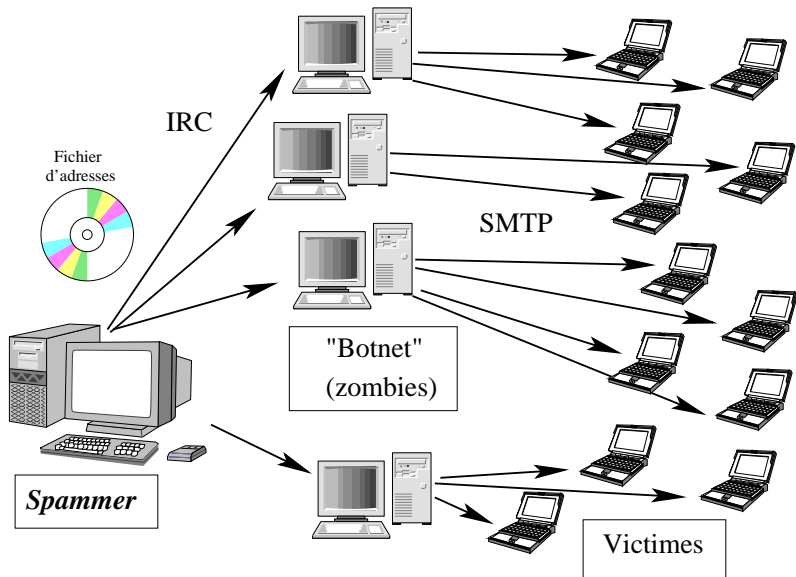
## Fonctionnement du *spam*

- ① Les *spammeurs* sous-traitent, comme dans le BTP,
- ② Ils utilisent un réseau de zombies. *A botnet is comparable to compulsory military service for Windows boxes*
- ③ Ils sont compétents techniquement mais sont pragmatiques : ils ne respectent les RFC que si ça les arrange,

## Fonctionnement du *spam*

- 1 Les *spammeurs* sous-traitent, comme dans le BTP,
- 2 Ils utilisent un réseau de zombies. *A botnet is comparable to compulsory military service for Windows boxes*
- 3 Ils sont compétents techniquement mais sont pragmatiques : ils ne respectent les RFC que si ça les arrange,
- 4 Ils gagnent de l'argent avec les 5 % de destinataires qui achètent (et donc ont tendance à arroser le plus possible).

# HOWTO spam





- 1 La solution parfaite n'existe pas,

- 1 La solution parfaite n'existe pas,
- 2 Toute solution a un coût,

- 1 La solution parfaite n'existe pas,
- 2 Toute solution a un coût,
- 3 Parmi les coûts, les **faux positifs** ou dommages collatéraux.

- ① La solution parfaite n'existe pas,
- ② Toute solution a un coût,
- ③ Parmi les coûts, les **faux positifs** ou dommages collatéraux.
- ④ Autre coût important : le temps humain et les ressources machines.

Postfix se contrôle par :

Postfix se contrôle par :

- `main.cf` : les options,

Postfix se contrôle par :

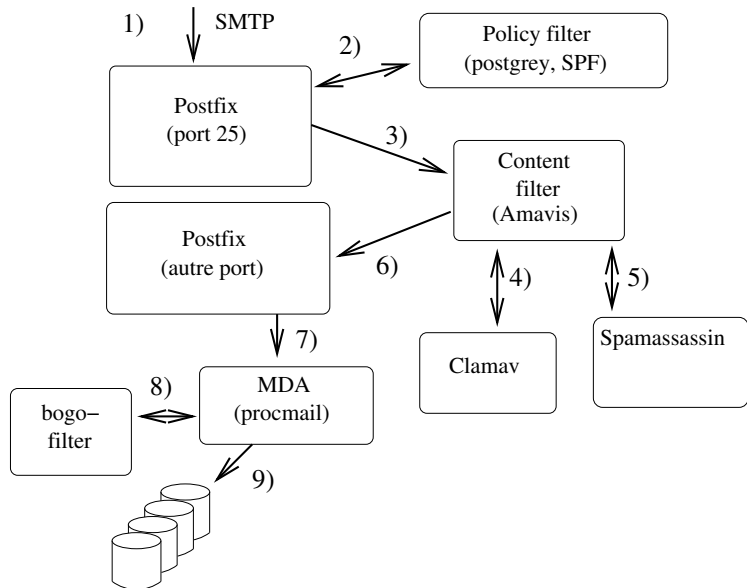
- `main.cf` : les options,
- `master.cf` : les démons.

Postfix se contrôle par :

- `main.cf` : les options,
- `master.cf` : les démons.

Il peut sous-traiter du travail à un programme comme Amavis.

# Le chemin du courrier dans votre Postfix



## Exemple de main.cf

À chaque étape du traitement SMTP, on peut tester (les tests sont séparés par des virgules et le premier qui donne un résultat non-nul termine la liste).

```
smtpd_helo_restrictions = permit_mynetworks, reject_invalid_hostname
smtpd_sender_restrictions = reject_unknown_sender_domain,
                           check_sender_mx_access cidr:/etc/postfix/badmxcidr
smtpd_recipient_restrictions = permit_mynetworks,
                              reject_unauth_destination,
                              check_policy_service unix:private/policy,
                              check_policy_service inet:127.0.0.1:60000
```

## Exemple de master.cf

C'est un peu l'inetd.conf ou le inittab de Postfix.

```
# service type private unpriv chroot wakeup maxproc command + args
#
# =====
smtp inet n - - - - smtpd
pickup fifo n - - 60 1 pickup
cleanup unix n - - - 0 cleanup
```

## Postfix *policy filter*

Ce filtre reçoit de Postfix un certain nombre de données :

```
helo_name=some.domain.example  
sender=foo@example.org  
recipient=bar@foo.example  
client_address=10.2.3.4  
...
```

et il peut répondre OK, REJECT, DISCARD, DUNNO, etc.

## Postfix *policy filter*

Ce filtre reçoit de Postfix un certain nombre de données :

```
helo_name=some.domain.example
sender=foo@example.org
recipient=bar@foo.example
client_address=10.2.3.4
...
```

et il peut répondre OK, REJECT, DISCARD, DUNNO, etc.

### Le *Policy Filter*

permet d'étendre très simplement Postfix, dans le langage de programmation de son choix.

## Postfix *policy filter*

Ce filtre reçoit de Postfix un certain nombre de données :

```
helo_name=some.domain.example
sender=foo@example.org
recipient=bar@foo.example
client_address=10.2.3.4
...
```

et il peut répondre OK, REJECT, DISCARD, DUNNO, etc.

### Le *Policy Filter*

permet d'étendre très simplement Postfix, dans le langage de programmation de son choix.

Principale limite : il ne transmet pas le corps du message.

Les *content filter* sont des serveurs SMTP et reçoivent donc tout le message.

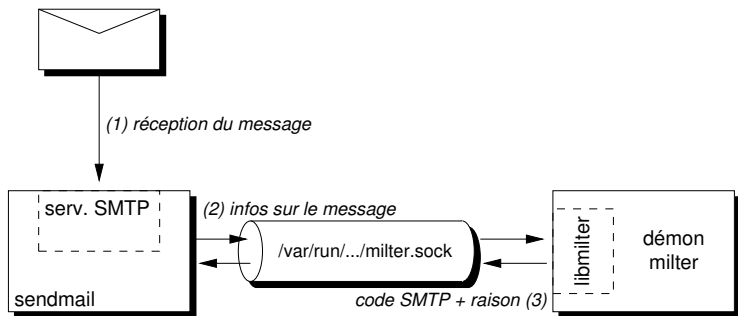
Mais ils s'installent **après** la session SMTP du MTA de bordure et ne peuvent donc pas signaler au client extérieur un code d'erreur.

L'API *milter* est le principal outil sendmail pour la lutte anti-*spam*.

- depuis la version Sendmail 8.10 (mars 2000)
- mécanisme d'extension
- simple et souple, bien intégré à Sendmail
- programmable en C, mais extensions vers Perl, Tcl, Python, etc.
- très grand nombre de milters (domaine public et semi-commerciaux)

# Sendmail – Milter

Architecture générale d'un *milter* :



L'API Milter est définie par la librairie `libmilter.a`.  
Quelques exemples de fonctions de l'API :

Exemples de fonctions fournies par <code>libmilter.a</code> :	
<code>smfi_main()</code>	fonction principale du <i>milter</i>
<code>smfi_getsymval()</code>	récupère la valeur d'une macro Sendmail
<code>smfi_addheader()</code>	ajoute une en-tête au message
Exemples de fonctions à écrire (appelées par <code>smfi_main()</code> ) :	
<code>toto_envfrom()</code>	appelé après le MAIL FROM (enveloppe RFC 2821)
<code>toto_header()</code>	appelé après la réception d'une en-tête RFC 2822
<code>toto_body()</code>	traite ou altère le corps du message
Codes de retour définis :	
SMFIS_CONTINUE, _REJECT, _DISCARD, _ACCEPT et _TEMPFAIL	

Exemples de *milters* :

- Milter non liés au Spam :
  - milter-length : limite la taille des messages par adresse IP, par domaine, par émetteur, etc.
  - roundhouse : duplique les messages vers plusieurs serveurs (debug, migrations, etc.)
  - ... et beaucoup d'autres
- Milters liés au Spam :
  - j-chkmail (José-Marcio Martins da Cruz)
  - ... et beaucoup d'autres
- Milters généralistes :
  - MIME-Defang : configurable en Perl
  - tclmilter : accès à l'API en Tcl
  - ... et beaucoup d'autres

Site de référence : <http://www.milter.org/>

Introduction : <http://milter.free.fr/intro/> (Stéphane Lentz)

Possibilités offertes par l'architecture :

- un même `sendmail` peut appeler plusieurs *milters* à la suite
- utilisation d'une connexion TCP au lieu d'une *socket* locale  
⇒ architectures intéressantes pour un MTA central (un seul frontal `sendmail`, plusieurs serveurs spécialisés pour les *milters*)

Possibilité d'avoir plusieurs *milters* en séquence. Par exemple :

- greylisting
- antivirus
- spamassassin
- etc

Configuration d'un milter :

- fichier de configuration du *milter*  
⇒ dépend de chaque *milter*
- fichier `sendmail.cf` pour prise en compte par `sendmail`
  - définition du mode de communication avec le *milter*
  - définition des informations envoyées au *milter*
  - activation du *milter*

Modification du `sendmail.cf` pour définir le mode de communication avec les *milters* :

```
X toto,Socket=local:/var/run/toto.sock, Flags=T, Timeouts=S:4m;R:4m
```

- nom interne (au fichier `sendmail.cf`) du *milter*
- définition de la *socket* ou de la connexion TCP
- flags en cas de non réponse du *milter* :
  - rien : faire comme si le *milter* répondait
  - T : erreur temporaire, ré-essayer
  - R : rejet de la connexion
- délais maximum d'attente
  - ici : 4 minutes pour envoyer une requête ou lire une réponse sur la *socket*

## Sendmail – Milter

Modification du `sendmail.cf` pour définir les informations communiquées aux *milters* :

```
0 Milter.macros.connect=b,j,{if_addr},{daemon_name},{if_name}
0 Milter.macros.envfrom=i
0 Milter.macros.envrcpt={rcpt_addr}
```

- définition du contexte :
  - `connect` : lors de la connexion (cf fonction `toto_connect()`)
  - `envfrom` : réception du MAIL FROM (cf `toto_envfrom()`)
  - etc.
- définition des informations : macros (cf doc de `sendmail`)
  - `j` : nom complet de la machine
  - `if_addr` : l'adresse IP sur laquelle a été reçu le message
  - `i` : l'id du message
  - etc.

⇒ on n'invente pas ces informations!

⇒ elles sont (devrait être) fournies dans la documentation du *milter*

Modification du `sendmail.cf` pour activer les *milters* :

```
0 InputMailFilters=toto,titi
```

- le message est fourni séquentiellement à tous les *milters*
- arrêt du parcours au premier qui échoue

Principe : retarder délibérément le courrier en envoyant un code d'erreur temporaire (4xx).

Les *spammeurs* actuels ne réessaient pas.

- 1 Au début, tout le monde est inconnu,
- 2 Un nouveau triplet <adresse IP, expéditeur, destinataire> arrive : il est mis en liste grise pour N minutes,
- 3 **S'il réessaie après P minutes, il est mis dans la liste blanche pour Q minutes,**
- 4 S'il ne réessaie pas, il est supprimé de la liste grise au bout des N minutes,
- 5 Si le triplet continue à être actif, sa durée de séjour (Q minutes) dans la liste blanche est prolongée,
- 6 En cas d'inactivité, il est supprimé de la liste blanche au bout de Q minutes.

On peut aussi ajouter manuellement des adresses dans la liste blanche.

- les clients réguliers ne sont donc pas retardés
- pas de faux positifs
- quelques rares clients SMTP non conformes  $\Rightarrow$  mise en liste blanche
- efficace  $\Rightarrow$  exemple sur le réseau métropolitain strasbourgeois Osiris
  - mis en œuvre en juin 2004
  - 300 000 msg/jour  $\rightarrow$  80 000 msg/jour
  - soulagement des serveurs

## *greylisting* avec Postfix

Postfix : on utilise Postgrey

main.cf :

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

En cas de problème, on peut utiliser une liste blanche :  
`/etc/postgrey/whitelist_clients`.

Milter-greylis : <http://hcpnet.free.fr/milter-greylis/>

- auteur : Emmanuel Dreyfus
- développement actif
- support d'IPv6, de SPF
- efficacité : maintien des listes grise et blanche en mémoire
- sûreté : recopie périodique sur disque (simple fichier texte)
- deux modes :
  - classique : basé sur le triplet « IP source, expéditeur, destinataire »
  - paresseux : basé seulement sur l'adresse IP source
- support de fermes de serveur

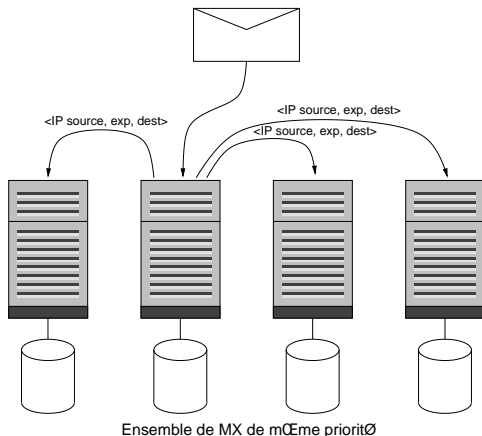
⇒ très complet et très fiable

## greylisting avec sendmail

Fichier de configuration : greylist.conf

```
peer 130.79.200.151          # la ferme de serveurs
peer 2001:660:2402::175     # la ferme de serveurs
acl whitelist addr 127.0.0.0/8 # adresse locale IPv4
acl whitelist addr ::1/128   # adresse locale IPv6
acl whitelist addr 130.79.0.0/16 # adresses autorisees IPv4
acl whitelist addr 2001:660:2402::/48 # adresses autorisees IPv6
lazyaw                      # mode paresseux
dumpfreq 10m                # base en memoire -> disque
timeout 5d                  # sortie de la liste grise
noaccessdb                  # obligatoire
greylist 1h                 # laisser rentrer apres 1h
autowhite 5d                # sortie de la liste blanche
nospf                       # pas d'utilisation de SPF
# Liste des clients SMTP non conformes
acl whitelist addr 12.5.136.141/32 # Southwest Airlines
```

Constitution d'une ferme de serveurs de liste grise :



- ensemble de serveurs de même priorité (MX équivalents)
- diffusion des listes grise et blanche en temps réel

# Filtre à heuristique

Principe : regarder des “marqueurs de spamicité” dans le message.

Exemples (tests de SpamAssassin) :

# Filtre à heuristique

Principe : regarder des “marqueurs de spamicité” dans le message.

Exemples (tests de SpamAssassin) :

- HTML link text says "opt out" or similar

# Filtre à heuristique

Principe : regarder des “marqueurs de spamicité” dans le message.

Exemples (tests de SpamAssassin) :

- HTML link text says "opt out" or similar
- HTML has "bgsound" tag

# Filtre à heuristique

Principe : regarder des “marqueurs de spamicité” dans le message.

Exemples (tests de SpamAssassin) :

- HTML link text says "opt out" or similar
- HTML has "bgsound" tag
- CGI in .biz TLD other than third-level "www"

# Filtre à heuristique

Principe : regarder des “marqueurs de spamicité” dans le message.

Exemples (tests de SpamAssassin) :

- HTML link text says "opt out" or similar
- HTML has "bgsound" tag
- CGI in .biz TLD other than third-level "www"
- Message talks about enhancing men

# Filtre à heuristique

Principe : regarder des “marqueurs de spamicité” dans le message.

Exemples (tests de SpamAssassin) :

- HTML link text says "opt out" or similar
- HTML has "bgsound" tag
- CGI in .biz TLD other than third-level "www"
- Message talks about enhancing men

Pris séparément, aucun de ces tests n'est suffisant

Le bon filtre à heuristiques les combine (et les scores sont calculés **automatiquement**, pour éviter tout biais humain).

# Configurer SpamAssassin

Supprimer un test :

```
# Test rfc-ignorant, lent et dangereux  
score RCVD_IN_RFCI 0
```

Mettre un expéditeur en liste blanche :

```
whitelist_from jacques@example.org
```

# SpamAssassin avec Postfix

master.cf :

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

# SpamAssassin avec sendmail

Utilisation de spamass-milter :

<http://savannah.gnu.org/projects/spamass-milter/>

Pas de configuration spécifique du *milter*

⇒ uniquement configuration de SpamAssassin lui-même.

Le *milter* ne fait que mettre en commun l'appel au client du démon spamd ⇒ efficacité

Principe : on soumet au filtre un corpus de *spam* et un de *ham*.

Au bout d'un moment, le filtre a identifié le vocabulaire du *spam*.

Principe : on soumet au filtre un corpus de *spam* et un de *ham*.

Au bout d'un moment, le filtre a identifié le vocabulaire du *spam*.

La combinaison des "scores" des mots utilise les statistiques bayésiennes (pour éviter qu'un seul "viagra" ne bloque un message).

# Base des mots du bayésien

Analyse d'un message avec bogofilter -v :

	n	pgood	pbad	fw
"\$99"	8	0.000275	0.002554	0.901823

pbad = "spamicité". fw = probabilité d'être un *spam*. La chaîne de caractères 99\$ est un bon marqueur de *spam*.

On peut aussi afficher toute la base de bogofilter avec bogoutil :

```
Viagra 88 0 20041116
```

Viagra est apparu dans 88 *spams* et aucun *ham*.

# Entrainement de bogofilter

```
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
```

# bogofilter avec Postfix

```
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
```

# bogofilter avec sendmail

Utilisation de miler-bogom :

<http://www.usebox.net/jjm/bogom/>

- appelle bogofilter  
⇒ le *milter* ne fait que faciliter l'intégration avec sendmail
- bonne flexibilité de configuration  
fichier bogom.conf
- quelques défauts d'implémentation  
n'utilise pas le répertoire par défaut de l'utilisateur non privilégié
- possibilité de mettre à jour la base en temps réel en fonction des messages reconnus  
⇒ la base peut croître considérablement  
⇒ usage limité à de faibles trafics

# bogofilter avec sendmail

Fichier de configuration : bogom.conf

```
policy pass           # en cas de spam : pass/reject/discard
subject_tag "[SPAM] " # marquage du sujet
training 0            # n'actualise pas la base
body_limit 1m        # pas d'analyse au del^e0
```

Le programme bogofilter ajoute sa propre en-tête (même en cas de marquage du sujet) :

```
X-Bogosity: Ham, tests=bogofilter, spamicity=...
```

# Authentification

Une famille de protocoles qui vise à **authentifier** l'émetteur. SMTP ne le fait pas par défaut (pour de très bonnes raisons).

# Authentification

Une famille de protocoles qui vise à **authentifier** l'émetteur. SMTP ne le fait pas par défaut (pour de très bonnes raisons).

## Attention

**Authentifier** n'est pas **autoriser**. Mon passeport prouve que je m'appelle bien Jean Dupont (authentification), pas que je suis honnête ou digne de parler à JRES (autorisation).

# Authentification

Une famille de protocoles qui vise à **authentifier** l'émetteur. SMTP ne le fait pas par défaut (pour de très bonnes raisons).

## Attention

**Authentifier** n'est pas **autoriser**. Mon passeport prouve que je m'appelle bien Jean Dupont (authentification), pas que je suis honnête ou digne de parler à JRES (autorisation).

L'authentification permet des listes blanches, rend d'avantage responsable. . . Indirectement, l'authentification peut donc aider à la lutte contre le *spam*.

## Sender Policy Framework

Authentifier via une déclaration dans le DNS des serveurs autorisés.

```
% dig TXT freebsd.org.  
... v=spf1 ip4:216.136.204.119 ~all
```

Ici, 216.136.204.119 est seul autorisé.

## Sender Policy Framework

Authentifier via une déclaration dans le DNS des serveurs autorisés.

```
% dig TXT freebsd.org.  
... v=spf1 ip4:216.136.204.119 ~all
```

Ici, 216.136.204.119 est seul autorisé.

### Identité testée

SPF teste le domaine de l'expéditeur indiqué dans l'**enveloppe**  
(MAIL FROM du RFC 2821)

Authentification du domaine par signature cryptographique.

Exposé plus détaillé à JRES.

On utilise le *policy filter*.

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

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

Trois voies possibles d'utilisation de SPF :

- 1 mettre automatiquement les tentatives authentifiées dans la liste blanche du *greylisting*  
⇒milter-grey-list
- 2 marquer et éventuellement rejeter les messages dont l'authentification SPF échoue  
⇒sid-milter
- 3 ajouter l'authentification SPF dans les critères de pondération de reconnaissance de *spam*  
⇒spamass-milter

Première solution : mettre automatiquement les tentatives authentifiées dans la liste blanche du *greylisting* :

- compilation de milter-greylst avec le support SPF
- nécessite une version  $\geq 2.0$  de milter-greylst
- suppression du mot-clef `nospf` dans `gerylist.conf`

⇒ facilite la vie des *spammeurs* (les *spams* rentrent plus vite)

# SPF avec sendmail

Deuxième solution : marquer et éventuellement rejeter les messages dont l'authentification SPF échoue

⇒ Utilisation de sid-milter : <http://sendmail.net/sid-milter/>

- par les auteurs de sendmail
- version très préliminaire (0.2 actuellement)
- supporte également l'authentification Sender-Id
- configurable sur la ligne de commande
- marquage, et éventuellement rejet des tentatives non authentifiées  
⇒ par SPF et/ou Sender-Id

Troisième solution : ajouter l'authentification SPF dans les critères de pondération de reconnaissance de *spam*, et en particulier les filtrages heuristiques de type SpamAssassin.

⇒laisser faire spamass-milter et SpamAssassin

⇒solution recommandée si vous voulez utiliser SPF

# Test de l'existence de l'émetteur

On contrôle que le domaine de l'émetteur existe.

Idéalement, on contrôle aussi que le MX est raisonnable (ne pointe pas vers ".", vers 127.0.0.1, ...).

Postfix : `reject_unknown_sender_domain`

Sendmail : règles anti-spam du kit Jussieu

## Techniques déconseillées ou trop récentes

Il y a aussi beaucoup de techniques dont les effets négatifs sont connus et qui sont donc **déconseillées**.

Ou bien qui sont trop récentes ou trop marginales pour être vraiment recommandées aujourd'hui.

## Principe

Lister des adresses IP publiquement, par exemple dans le DNS. Le MTA peut décider de refuser le courrier de ces adresses.

## Principe

Lister des adresses IP publiquement, par exemple dans le DNS. Le MTA peut décider de refuser le courrier de ces adresses.

## Problème

Peu de listes sont gérées franchement et correctement : critères inconnus, retrait très lent.

## Principe

Demander confirmation au supposé expéditeur.

## Principe

Demander confirmation au supposé expéditeur.

## Problème

Reporte le coût de la lutte anti-*spam* sur les expéditeurs.

## Et ensuite ?

Nous avons vu un ensemble de **tests**. Chaque test est une **assertion** sur un message qui dit "C'est un *spam*" ou bien "Ce n'est pas un *spam*".

Comment synthétiser le résultat de tous les tests qu'on utilise ?

## Et ensuite ?

Nous avons vu un ensemble de **tests**. Chaque test est une **assertion** sur un message qui dit “C’est un *spam*” ou bien “Ce n’est pas un *spam*”.

Comment synthétiser le résultat de tous les tests qu’on utilise ?

- 1 Rejet : jeter immédiatement,

## Et ensuite ?

Nous avons vu un ensemble de **tests**. Chaque test est une **assertion** sur un message qui dit “C’est un *spam*” ou bien “Ce n’est pas un *spam*”.

Comment synthétiser le résultat de tous les tests qu’on utilise ?

- 1 Rejet : jeter immédiatement,
- 2 Privilège : laisser passer les bons,

## Et ensuite ?

Nous avons vu un ensemble de **tests**. Chaque test est une **assertion** sur un message qui dit “C’est un *spam*” ou bien “Ce n’est pas un *spam*”.

Comment synthétiser le résultat de tous les tests qu’on utilise ?

- 1 Rejet : jeter immédiatement,
- 2 Privilège : laisser passer les bons,
- 3 *Scoring* : combiner. C’est ce que fait SpamAssassin par défaut.

## Que faire ensuite ?

Une fois qu'on a décidé qu'un message était du *spam*, on peut :

## Que faire ensuite ?

Une fois qu'on a décidé qu'un message était du *spam*, on peut :

- 1 Si on est encore dans la session SMTP, envoyer une erreur 5xx : c'est sans doute la meilleure solution.

## Que faire ensuite ?

Une fois qu'on a décidé qu'un message était du *spam*, on peut :

- 1 Si on est encore dans la session SMTP, envoyer une erreur 5xx : c'est sans doute la meilleure solution.
- 2 Prévenir l'expéditeur (par un DSN, *Delivery Status Notification*). **C'était** la meilleure solution mais aujourd'hui, avec les usurpations, il vaut mieux :

## Que faire ensuite ?

Une fois qu'on a décidé qu'un message était du *spam*, on peut :

- 1 Si on est encore dans la session SMTP, envoyer une erreur 5xx : c'est sans doute la meilleure solution.
- 2 Prévenir l'expéditeur (par un DSN, *Delivery Status Notification*). **C'était** la meilleure solution mais aujourd'hui, avec les usurpations, il vaut mieux :
- 3 Jeter le message. Parfait pour les *spams* et les virus mais attention à la traçabilité.

## Que faire ensuite ?

Une fois qu'on a décidé qu'un message était du *spam*, on peut :

- 1 Si on est encore dans la session SMTP, envoyer une erreur 5xx : c'est sans doute la meilleure solution.
- 2 Prévenir l'expéditeur (par un DSN, *Delivery Status Notification*). **C'était** la meilleure solution mais aujourd'hui, avec les usurpations, il vaut mieux :
- 3 Jeter le message. Parfait pour les *spams* et les virus mais attention à la traçabilité.
- 4 Marquer le message (en-tête spécifique) et laisser le MDA ou le MUA trier.

Le *policy filter* et les directives `reject_*` permettent d'envoyer une erreur SMTP.

*Policy filter* : REJECT est définitif, DEFER\_IF\_PERMIT ou DEFER\_IF\_REJECT temporaire. On peut aussi mettre à la place un code numérique (bien lire le RFC 2821 avant).

## Dans la session SMTP de Postfix

Le *policy filter* et les directives `reject_*` permettent d'envoyer une erreur SMTP.

*Policy filter* : REJECT est définitif, DEFER\_IF\_PERMIT ou DEFER\_IF\_REJECT temporaire. On peut aussi mettre à la place un code numérique (bien lire le RFC 2821 avant).

On peut aussi, dans `main.cf`, changer les codes SMTP renvoyés (`postconf | grep code` pour les voir tous.)





## Marquer les messages

C'est la solution **recommandée**. Le MDA ou le MUA peut trier ensuite.

Avec Sieve (GNU mailutils ou Cyrus) :

```
if header :contains "X-Spam-Flag" "YES" {  
  fileinto "INBOX.Junk";  
}
```

Avec maildrop :

```
if (/^X-Spam-Flag:.*YES/)  
{  
    to $DEFAULT/.spam  
}
```

## Un petit mot sur Amavis

Il dépieute les parties MIME d'un message, peut effectuer certains tests sur ces parties et appeler des programmes de tests comme SpamAssassin ou Clamav (anti-virus).

```
### http://www.clamav.net/ - backs up clamd or Mail::ClamAV  
['ClamAV-clamscan', 'clamscan',  
  "--stdout --disable-summary -r --tempdir=$TEMPBASE {}]", [0], [1],  
qr/^.*?: (?!Infected Archive)(.*) FOUND$/ ],
```

# En conclusion

## En conclusion

- 1 Choisissez un **petit** nombre de techniques,

## En conclusion

- 1 Choisissez un **petit** nombre de techniques,
- 2 Testez-les avant déploiement effectif (par exemple en se contentant de marquer ou de *loguer*),

## En conclusion

- 1 Choisissez un **petit** nombre de techniques,
- 2 Testez-les avant déploiement effectif (par exemple en se contentant de marquer ou de *loguer*),
- 3 Communiquez vos tests à l'extérieur et à vos utilisateurs.