Postfix Anti-Spam Workshop

Ralf Hildebrandt ©2007

state-of-mind

LISA'07
Dallas, November 2007

Which methods are available?

In order to prevent spam from reaching the users' mailboxes there are basically four methods:

- syntactic checks
  Check if the data is syntactically valid
- plausibility checks
  Check if the data is plausible
- reputation
  Reject mail from hosts associated with "bad" networks
  Accept mail from sender domains known to be "good"
- content analysis
  Look at the content of the mail

advantages and disadvantages

pro:

- really safe
- fast (in comparison to content analysis), thus cheap to deploy

contra:

- easy to circumvent

advantages and disadvantages

contra:

- one bad apple spoils the whole lot
- whom to trust?
- how is the reputation built up?
- We’re the good guys now, we **used** to be bad, but now we’re good
- the sender address may be forged

pro:

- fast (in comparison to content analysis), thus cheap to deploy

- In-depth analysis is s l o o o o o o w
- spammers “optimize” their spam to actually pass SpamAssassin
- you could write your own rules, though

Postfix can use all three approaches, using different implementations:
Built-in:

- `smtpd_*_restrictions`
- `header_checks`
- `mime_header_checks`
- `body_checks`

Calling external programs:

- policy-delegation (postgrey, policyd-weight, policyd, ...)
- `content_filter` (amavisd-new)
- `smtpd_proxy_filter` (clamsmtpd)
- `smtpd_milters` (badly coded stuff for sendmail you cannot live without)

- The built-in mechanisms are fast, but restricted in what they can do. But they can be deployed easily, usually with great success.
- The external programs can do virtually anything with the mail, but somebody has to code these features into the external programs.

We'll jump into the configuration section directly, explaining the groundworks as we go.

Configuration

We'll configure an anti-spam relay step-by-step, explain the rationale behind our design decisions and discuss the pros and cons (cost & efficiency) of the different methods:

- built-in restrictions
- policy-delegation (policyd-weight)
- `content_filter` (amavisd-new with SpamAssassin)
- `smtpd_proxy_filter` (clamsmtp)
- `smtpd_milters` (DKIM milter)

built-in restrictions

Postfix comes with a default set of `smtpd_*_restrictions`:

```
smtpd_recipient_restrictions =
   permit_mynetworks,
   reject_unauth_destination
```

That setup prevents relaying and (implicitly) receiving mail for non-existing recipient-addresses on your system.

How the restrictions work

The restrictions are read and applied from left to right, a line starting with withspace continues the last line.

- `permit_mynetworks`: allows the transaction if the client is listed in `mynetworks`. Returns `DUNNO` otherwise. Upon `DUNNO`, postfix continues with the next restriction.
- `reject_unauth_destination`
  reject the transaction if it would constitute relaying. Returns `DUNNO` otherwise.
- So Postfix continues, and now Postfix implies `reject_unlisted_recipient` at the end – it rejects mail to unlisted recipients. Returns `DUNNO` otherwise.
- And now, at the very end, a `permit` is implied.

It is time to pimp ze server

Remember the four approaches? Syntax and plausibility vs. reputation vs. content analysis? We'll stick with syntax and plausibility for now, it's easier to implement.

```
smtpd_recipient_restrictions =
  reject_non_fqdn_sender
  reject_unknown_sender_domain,
  reject_non_fqdn_recipient
  reject_unknown_recipient_domain
  permit_mynetworks,
  reject_unauth_destination
```


Explanation

We added `reject_non_fqdn_sender` and `reject_unknown_sender_domain` – and the equivalent for the recipient addresses: `reject_non_fqdn_recipient` and `reject_unknown_recipient_domain`

- `reject_non_fqdn_sender` / `reject_non_fqdn_recipient` require everybody to use the form `sender@fqdn` instead of `sender`.
- `reject_unknown_sender_domain` / `reject_unknown_recipient_domain` perform DNS lookups to check if the sender / recipient domains exist.

All these checks are made before `permit_mynetworks`, because I don't like it when my users send out crappy mail.

Setting error codes appropriately

```
unknown_address_reject_code   = 554
unknown_hostname_reject_code  = 554
unknown_client_reject_code = 550
unverified_sender_reject_code = 554
```


Identity crisis (plausibility)

```
smtpd_helo_required = yes

smtpd_recipient_restrictions =
  reject_non_fqdn_sender
  reject_unknown_sender_domain,
  reject_non_fqdn_recipient
  reject_unknown_recipient_domain
  permit_mynetworks,
  reject_unauth_destination
  check_helo_access \
     pcre:/etc/postfix/helo_checks.pcre
```

Every server needs to send a HELO/EHLO greeting (enforced by `smtpd_helo_required = yes`). What we do now is to check that greeting for plausibility.
This is done using:
`check_helo_access`
`pcre:/etc/postfix/helo_checks.pcre`
Postfix queries a PCRE (perl Compatible Regular Expression) type map with the HELO/EHLO argument as key.
Let's have a look, shall we?

HELO/EHLO patterns

```
/^localhost$/           550 Don't use localhost
/^(base\.)?deroo\.net$/ 550 Don't use my own domain/hostname
/^\[?67\.130\.3\.42\]?$/
   550 Spammer comes to me \
       Greets me with my own IP \
       His mail I shall not see.
/^[0-9.-]+$/
   550 Your software is not RFC 2821 compliant: \
       EHLO/HELO must be a domain or an \
       address-literal (IP enclosed in brackets)
```

HELO/EHLO patterns explanation

This example assumes that my server's hostname is base.deroo.net and that its IP is 67.130.3.42
So, if some client greets us with `base.deroo.net`, then it's **lying**. Likewise, that client is probably not `localhost`, so why does it say so?
Note that this will **not** block clients from `mynetworks`, since these are already being permitted by `permit_mynetworks` **before** our `check_helo_access` could reject their mail.

More HELO/EHLO checks (plausibility and syntax)

- `reject_invalid_helo_hostname`
  Reject the request when the HELO or EHLO hostname syntax is invalid.
- `reject_non_fqdn_helo_hostname`
  Reject the request when the HELO or EHLO hostname is not in fully-qualified domain form, as required by the RFC.

“Real” reputation based decisions: DNSBLs

```
smtpd_recipient_restrictions =
  reject_non_fqdn_sender
  reject_unknown_sender_domain,
  reject_non_fqdn_recipient
  reject_unknown_recipient_domain
  permit_mynetworks,
  reject_unauth_destination
  reject_invalid_helo_hostname
  reject_non_fqdn_helo_hostname
  check_helo_access pcre:/etc/postfix/helo_checks.pcre
  reject_rbl_client zen.spamhaus.org
```

Postfix Anti-Spam Workshop
built-in
DNSBLs

`reject_rbl_client zen.spamhaus.org` would reject the transaction if the client was found in the RBL called `zen.spamhaus.org`

```
postfix/smtpd[14555]: NOQUEUE: reject:  RCPT from
unknown[124.197.160.191]: 554 5.7.1 Service unavailable;
Client host [124.197.160.191] blocked using zen.spamhaus.org;
http://www.spamhaus.org/query/bl?ip=124.197.160.191;
from=<pckitty@larrylink.com> to=<austin@deroo.net> proto=ESMTP
helo=<[124.197.160.191]>
```

But what if I want mail from these clients anyway?

Make an exception!

```
smtpd_recipient_restrictions =
  reject_non_fqdn_sender
  reject_unknown_sender_domain,
  reject_non_fqdn_recipient
  reject_unknown_recipient_domain
  permit_mynetworks,
  reject_unauth_destination
  reject_invalid_helo_hostname
  reject_non_fqdn_helo_hostname
  check_helo_access pcre:/etc/postfix/helo_checks.pcre
  check_sender_access hash:/etc/postfix/senders
  check_client_access hash:/etc/postfix/clients
  check_recipient_access hash:/etc/postfix/recipients
  reject_rbl_client zen.spamhaus.org
```

Postfix Anti-Spam Workshop
built-in
Exceptions

So, if there's a particular sender address you want to receive, regardless whether the IP is listed in `zen.spamhaus.org`, you fill in

```
my_isp_sux@kommkast.nett    OK
```

and

```
check_sender_access hash:/etc/postfix/senders
```

would return `OK` for that sender address, thus allowing the transaction.
A con of that approach may be that the spammer could simply send all his spam using that sender address.

Or, if you want to make an exception for an **client**-based blocking criterion, you could make the exception by **client**, thus using:

```
check_client_access hash:/etc/postfix/clients
```

instead.

Picking the example from the log:

```
postfix/smtpd[14555]: NOQUEUE: reject: RCPT from
unknown[124.197.160.191]: 554 5.7.1 Service unavailable;
Client host [124.197.160.191] blocked using zen.spamhaus.org;
http://www.spamhaus.org/query/bl?ip=124.197.160.191;
from=<pckitty@larrylink.com> to=<austin@deroo.net>
proto=ESMTP helo=<[124.197.160.191]>
```

The exception:

```
124.197.160.191    OK
```

Adding more blacklists

```
smtpd_recipient_restrictions =
  reject_non_fqdn_sender
  reject_unknown_sender_domain,
  reject_non_fqdn_recipient
  reject_unknown_recipient_domain
  permit_mynetworks,
  reject_unauth_destination
  reject_invalid_helo_hostname
  reject_non_fqdn_helo_hostname
  check_helo_access pcre:/etc/postfix/helo_checks.pcre
  reject_rbl_client zen.spamhaus.org
  reject_rhsbl_sender dsn.rfc-ignorant.org
```

Postfix Anti-Spam Workshop
built-in
More DNSBLs

`reject_rhsbl_sender dsn.rfc-ignorant.org` will reject mail from systems that refuse to accept bounces (sent using the empty envelope sender, `MAIL FROM:<>`).
If you accept mail from those senders, and the mail turns out to be undeliverable (over quota, delivery problems, delay), your server will send a bounce.
That bounce won't be deliverable, thus it will end up as "double-bounce".
`2bounce_notice_recipient = postmaster`
That would probably be you :)
Again, you need the ability to whitelist certain senders.

Adding more tidbits

`strict_rfc821_envelopes = yes`
Require that addresses received in SMTP MAIL FROM and RCPT TO commands are enclosed with <>, and that those addresses do not contain RFC 822 style comments or phrases. This stops mail from poorly written software.
`disable_vrfy_command = yes`
Disable the `SMTP VRFY` command. This stops some techniques used to harvest email addresses.

Show me your MX and I know who you are

Sometimes, spammer use throw-away domains to send out their spam. The MX entries for these domains point back to their own networks (because they need to have control over the MX host).
We can use that fact for another type of check:

```
check_client_access    cidr:/etc/postfix/drop.cidr
check_sender_mx_access cidr:/etc/postfix/drop.cidr
check_sender_ns_access cidr:/etc/postfix/drop.cidr
```

`/etc/postfix/drop.cidr` contains a list of "bad networks". Any client, any sender having an MX or any sender having a NS (nameserver) in these nets is being rejected.

Where do I get drop.cidr from?

`/etc/postfix/drop.cidr` looks like this:

```
116.199.128.0/19 REJECT SBL56563
116.50.8.0/21    REJECT SBL54501
122.8.0.0/15     REJECT SBL52788
...
```

and is generated from the Spamhaus drop (Don’t Route Or Peer) list.

DROP

DROP (Don’t Route Or Peer) is an advisory "drop all traffic" list, consisting of stolen ’zombie’ netblocks and netblocks controlled entirely by professional spammers. DROP is a tiny sub-set of the SBL designed for use by firewalls and routing equipment.

```
wget -q -nd --output-document=- \
  http://www.spamhaus.org/drop/drop.lasso |
  awk ’/; SBL/ {printf("%s\tREJECT %s\n",$1,$3)}’ >
  /etc/postfix/drop.cidr
```

The resulting logs look like this:

```
postfix/smtpd[32241]: NOQUEUE: reject: RCPT from
81-208-83-246.fastres.net[81.208.83.246]: 554 5.7.1
<nuon657@matchedtween.net>: Sender address rejected:
SBL48941; from=<nuon657@matchedtween.net>
to=<renate.kirschner@charite.de> proto=ESMTP
helo=<[81.208.83.246]>

postfix/smtpd[11393]: NOQUEUE: reject: RCPT from
69-50-167-194.esthost.com[69.50.167.194]: 554 5.7.1
<69-50-167-194.esthost.com[69.50.167.194]>: Client host rejected:
SBL53320; from=<> to=<FCX9UBO8ZK@rrk-berlin.de> proto=SMTP
helo=<misteriozo.com>
```


And another MX check

```
check_sender_mx_access cidr:/etc/postfix/bogon_networks.cidr
```

`/etc/postfix/bogon_networks.cidr` looks familiar:

```
0.0.0.0/7   REJECT IP address of MX host is in bogon netspace 1
2.0.0.0/8   REJECT IP address of MX host is in bogon netspace 2
5.0.0.0/8   REJECT IP address of MX host is in bogon netspace 3
...
```

It's generated from the famous page `http://www.cymru.com/Documents/bogon-bn-agg.txt`
If an MX of a sender points to bogon netspace, things are amiss.

Some more RBLs to use

`reject_rbl_client list.dsbl.org`

DSBL is the Distributed Sender Blackhole List, it publishes the IP addresses of hosts which have sent special test email to listme@listme.dsbl.org or another listing address.

`reject_rbl_client ix.dnsbl.manitu.net`

The iX blacklist is made of automatically generated entries without distinguishing open proxies from relays, dialup gateways, and so on. An email source just has to send spam to us to make it on the list. After about four days the IP address will be removed if there is no new spam from there.

Sender address verification

A sender or recipient address is verified by probing the nearest MTA for that address, without actually delivering mail. The nearest MTA could be Postfix itself, or it could be a remote MTA (SMTP interruptus). Probe messages are like normal mail, except that they are never delivered, deferred or bounced; probe messages are always discarded.
It’s as easy as adding:

`reject_unverified_recipient`

to the bottom of the `smtpd_recipient_restrictions`. Put it at he end, because it’s expensive.

Expensive?

Some sites may blacklist you when you are probing them too often (a probe is an SMTP session that does not deliver mail), or when you are probing them too often for a non-existent address.
Unfortunately, some major sites such as YAHOO do not reject unknown addresses in reply to the RCPT TO command, but report a delivery failure in response to end of DATA after a message is transferred. Postfix address verification does not work with such sites.

Caching

```
address_verify_map = btree:/var/mta/verify
```

`/var/mta/verify.db` can get huge, make sure you have plenty of space.

```
# ls -lh verify.db
-rw-r--r-- 1 root root 224M 2007-09-16 11:17
             verify.db
```

Whoops!

Selective SAV I

Use sender address verification selectively!

```
check_sender_access hash:/etc/postfix/frequently_forged_senders
```

`/etc/postfix/frequently_forged_senders` looks like this:

```
abaforum.es           reject_unverified_recipient
abatron.de            reject_unverified_recipient
abbendon.demon.co.uk  reject_unverified_recipient
...
```


Selective SAV II

You could also perform the check if the mail is being delivered from a client that looks like it's a dialup:

```
check_sender_access pcre:/etc/postfix/dialups.pcre
```

`/etc/postfix/dialups.pcre` contains:

```
/\.rima-tde\.net$/ reject_unverified_recipient
/\.auna\.net$/     reject_unverified_recipient
/\.comcast\.net$/  reject_unverified_recipient
/\.uu\.net$/       reject_unverified_recipient
/\.adelphia\.net$/ reject_unverified_recipient
/\.attbi\.com$/    reject_unverified_recipient
/\.rr\.com$/       reject_unverified_recipient
/\.pacbell\.net$/  reject_unverified_recipient
...
```

If you want to hit your head against the wall, you can find SQLgrey's regexp (which is used to switch from class-C to whole IP based greylisting when a dialup is detected) here:
`http://sqlgrey.bouton.name/dyn_fqdn.regexp`
This is awfully long but replaced the more simple approach SQLgrey inherited from Postgrey (matching IP components in the fqdn) because users reported better dialup detection with it.
There's another for matching well-known SMTP name patterns here:
`http://sqlgrey.bouton.name/smtp_server.regexp`

But there's more built-in stuff!

We never actually looked at the contents of the mail with Postfix.

```
header_checks =
   pcre:/etc/postfix/header_checks.pcre
mime_header_checks =
   pcre:/etc/postfix/mime_header_checks.pcre
body_checks =
   pcre:/etc/postfix/body_checks.pcre
```

Postfix will apply those after the transaction has passed the `smtpd_*_restrictions` listed above.

What can those do - header_checks

We can use them to strip away the headers that show we're using amavisd-new:

```
/^Received: from localhost \(localhost \[127\.0\.0\.1\]\)/
   IGNORE
/^Received: from mail\.charite\.de \(\[127\.0\.0\.1\]\)/
   IGNORE
/^Received: from mail-ausfall\.charite\.de \(\[127\.0\.0\.1\]\)/
   IGNORE
/^Received: by mail\.charite\.de \(Postfix, from userid 0\)/
   IGNORE
```

You can also solve that problem where it arises - insert:

```
$insert_received_line = 0;
```

into `amavisd.conf`

What can those do - log the User-Agent

```
/^(User-Agent|X-Mailer): (.*)$/ WARN 0xdeadbeef $2
```

```
Sep 16 10:34:36 mail postfix/cleanup[18854]: 0FFDB1661AF: warning:
  header X-Mailer: Microsoft Outlook Express 6.00.2900.2180 from
  localhost[127.0.0.1]; from=<> to=<m.becker@charite.de> proto=ESMTP
  helo=<localhost>: 0xdeadbeef
  Microsoft Outlook Express 6.00.2900.2180
```

What can those do - eliminate crapware spam

```
/Date:.*([3-9].:..:|2[4-9]:..:|:[6-9].:|:..:6[1-9])/
   REJECT Invalid time "${1}..." in Date header

postfix/cleanup[496]: 10EDC165FE5: reject: header Date: , 16 Sep 2007
25:11:24 +0800 from moutng.kundenserver.de[212.227.126.179];
from=<mike@chiarabartoletti.com> to=<anja.elstner@charite.de>
proto=ESMTP helo=<moutng.kundenserver.de>: 5.7.1 Invalid time
"25:11:..." in Date header
```

Postfix Anti-Spam Workshop
built-in II
Sender address verification

What can those do - `mime_header_checks`

We can use `mime_header_checks` to reject mails containing attachments with certain unwanted extensions:

```
/name=\"(.*)\.(386|bat|chm|cpl|cmd|com|do|exe|hta|jse)\"$/
   REJECT Unwanted attachment $1.$2
/name=\"(.*)\.(pif|reg|rm|scr|shb|shm|shs|sys|vbe|vbs)\"$/
   REJECT Unwanted attachment $1.$2
/name=\"(.*)\.(lnk|msi|ole|vxd|xl|xsl)\"$/
   REJECT Unwanted attachment $1.$2
```

What can those do - problems

Problem: If the filename if KOI encoded instead of ASCII or ISO, then our patterns won't match. But in practice, it works well:

```
postfix/cleanup[28546]: 1A7E5165F92: reject: header Content-Type:
application/x-msdownload;??name="load'ka.exe" from
AReims-157-1-35-31.w86-208.abo.wanadoo.fr[86.208.202.31];
from=<york.kuehnle@charite.de> to=<york.kuehnle@charite.de>
proto=ESMTP helo=<AReims-157-1-28-218.w86-208.abo.wanadoo.fr>: 5.7.1
Unwanted attachment/Unerwuenschter Anhang load'ka.exe
```


Blocking backscatter using body_checks

These are my last resort against backscatter spam or large volumes of spam that somehow manage to pass all other tests:

```
/Eurasian Business Directory/ REJECT Scam 27.05.2007 RHI
/WKN A0MJ2U/                  REJECT Spam 02.08.2007 RHI
/ISIN CH0029095327/           REJECT Spam 02.08.2007 RHI
```

Again, the body of the mail may not be ASCII or ISO, but KOI-something or UTF-something, so these plain text patterns may not work.
Use them sparingly.

Calling external programs: policy delegation

If the sender is X and the recipient is Y and the tide is high, I want to reject the mail, unless the length of the mail is a Mersenne prime.
You cannot implement this using Postfix's built-in restrictions.
You may be able to do this in Exim (I guess).

- cons
  - doesn't get to see the content of the mail
- pros
  - fast
  - gets to see everything else

This is where the policy delegation comes into play:
As of version 2.1, Postfix can delegate policy decisions to an external server that runs outside Postfix.
The Postfix policy delegation protocol is really simple. The client (Postfix) request is a sequence of `name=value` attributes separated by newline, and is terminated by an empty line.
The server (Program that runs outside Postfix) reply is one `name=value` attribute and it, too, is terminated by an empty line.

Here is an example of all the attributes that the Postfix SMTP server sends in a delegated SMTPD access policy request:
Postfix version 2.1 and later:

```
request=smtpd_access_policy
protocol_state=RCPT
protocol_name=SMTP
helo_name=some.domain.tld
queue_id=8045F2AB23
sender=foo@bar.tld
recipient=bar@foo.tld
recipient_count=0
client_address=1.2.3.4
client_name=another.domain.tld
reverse_client_name=another.domain.tld
instance=123.456.7
```

Postfix version 2.2 and later:

```
sasl_method=plain
sasl_username=you
sasl_sender=
size=12345
ccert_subject=solaris9.porcupine.org
ccert_issuer=Wietse+20Venema
ccert_fingerprint=C2:9D:F4:87:71:73:73:D9:18:E7:C2:F3:C1:DA:6E:04
```

Postfix Anti-Spam Workshop
policy delegation
available attributes

Postfix version 2.3 and later:

```
encryption_protocol=TLSv1/SSLv3
encryption_cipher=DHE-RSA-AES256-SHA
encryption_keysize=256
etrn_domain=
[empty line]
```

The Postfix delegated policy client can connect to a TCP socket or to a UNIX-domain socket. Examples:

```
inet:127.0.0.1:9998
unix:/some/where/policy
unix:private/policy
```


policyd-weight

policyd-weight is a Perl policy daemon for the Postfix MTA (2.1 and later) intended to eliminate forged envelope senders and HELOs (e.g. in bogus mails). It allows you to score DNSBLs (RBL/RHSBL), `HELO`, `MAIL FROM` and client IP addresses before any queuing is done.
It allows you to `REJECT` messages which have a score higher than a certain threshold, providing improved blocking of spam and virus mails.
policyd-weight caches the most frequent client/sender combinations (spam as well as ham) to reduce the number of DNS queries.

To your `smtpd_recipient_restriction` you need to add:

```
smtpd_recipient_restriction =
   ...
   reject_unauth_destination
   ...
   check_policy_service inet:127.0.0.1:12525
```

That's it.

Postfix Anti-Spam Workshop
policy delegation
example: policyd-weight

There are many settings to tweak in policyd-weight, but let's look at the logs for now:

```
Sep 16 06:06:36 fatush postfix/policyd-weight[10901]: weighted
check: IN_DYN_NJABL=3.25 NOT_IN_SBL_XBL_SPAMHAUS=-1.5
IN_SPAMCOP=3.75 NOT_IN_BL_NJABL=-1.5 C L_IP_NE_HELO=8.5
RESOLVED_IP_IS_NOT_HELO=1.5 (check from: .joelgandara. - helo:
.[222.110.144.68]. - helo-domain: .68].)
FROM_NOT_FAILED_HELO(DOMAIN)=10 <client=222.110.144.68>
<helo=[222.110.144.68]> <from=patricia@joelgandara.com>
<to=webmaster@arschkrebs.de>, rate: 24

Sep 16 06:06:36 fatush postfix/policyd-weight[10901]: decided
action=550 Mail appeared to be SPAM or forged. Ask your
Mail/DNS-Administrator to correct HELO and DNS MX settings or
to get removed from DNSBLs; MTA helo: [222.110.144.68], MTA
hostname: unknown[222.110.144.68] (helo/hostname mismatch)
```

Pros and cons of amavisd-new

- cons
  - slow (mainly due to SpamAssassin!)
  - CPU intensive (mainly due to SpamAssassin!)
- pros
  - archiving is possible
  - per user settings
  - message is scanned once, instead of once per recipient
  - supports multiple virus scanners
  - can use spamassassin and/or dspam
  - lot of external packers

amavisd-new is immensely powerful

Postfix allows external programs access to the mails passing through after they have been queued.
An after-queue content filter receives unfiltered mail from Postfix (as described further below) and can do one of the following:

- Re-inject the mail back into Postfix, perhaps after changing content and/or destination.
- Discard or quarantine the mail.
- Reject the mail (by sending a suitable status code back to Postfix). Postfix will send the mail back to the sender address.

How does amavisd-new work?

- it receives mail via SMTP or LMTP
- it deconstructs the email
- it invokes multiple virus scanners on the parts and the whole
- it passes the mail through Mail::SpamAssassin
- it quarantines/passes/rejects the mail
- it adds some headers

How does the mail get into amavisd-new?

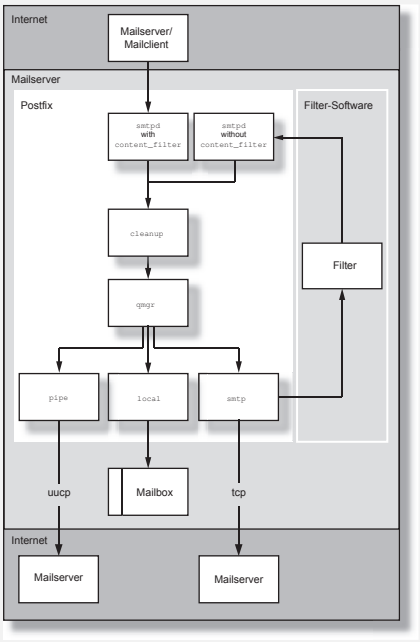

Figure: mail flow

main.cf

```
content_filter = scan:[localhost]:10025
```

Use the transport `scan` from `master.cf` to deliver the mail to `localhost`, port `10025`.

```
receive_override_options = no_address_mappings
```

Don't rewrite the recipient address. Pass it on to the `content_filter` unchanged.

So, all we need is amavisd-new to be listening on localhost, port 10025. And we need to define `scan` in `master.cf`.

```
# ==========================================================
# service type  private unpriv  chroot  wakeup  maxproc command
#               (yes)   (yes)   (yes)   (never) (100)
# ==========================================================
scan      unix  -       -       n       -       10      smtp
   -o smtp_send_xforward_command=yes
   -o disable_mime_output_conversion=yes
   -o smtp_generic_maps=
```

- With `smtp_send_xforward_command=yes`, the scan transport will try to forward the original client name and IP address through the content filter to the after-filter smtpd process, so that filtered mail is logged with the real client name IP address.
- The `disable_mime_output_conversion=yes` is a workaround that prevents the breaking of domainkeys and other digital signatures. This is needed because some SMTP-based content filters don't announce 8BITMIME support, even though they can handle 8-bit mail.
- The `smtp_generic_maps=` is a workaround that prevents local address rewriting with generic(5) maps. Such rewriting should happen only when mail is sent out to the Internet.

What next?

amavisd-new would not receive the mail from Postfix – but it needs a way back into Postfix. And that way must be exempt from being scanned again and again and again...

```
# ==========================================================================
# service       type  private unpriv  chroot  wakeup  maxproc command
#                     (yes)   (yes)   (yes)   (never) (100)
# ==========================================================================
localhost:10026 inet  n       -       n       -       10      smtpd
   -o content_filter=
   -o receive_override_options=no_unknown_recipient_checks,
      no_header_body_checks,no_milters
   -o smtpd_helo_restrictions=
   -o smtpd_client_restrictions=
   -o smtpd_sender_restrictions=
   -o smtpd_recipient_restrictions=permit_mynetworks,reject
   -o mynetworks=127.0.0.0/8
   -o smtpd_authorized_xforward_hosts=127.0.0.0/8
```

This way back into Postfix on `localhost:10026` has no checks whatsoever:

- The `content_filter=` overrides main.cf settings, and requests no content filtering for mail from the content filter. This is required or else mail will loop.
- We specify `no_unknown_recipient_checks` to disable attempts to find out if a recipient is unknown.
- We specify `no_header_body_checks` to disable header/body checks.
- We specify `no_milters` to disable Milter applications (this option is available only in Postfix 2.3 and later).

- We don't specify `no_address_mappings` here. This enables virtual alias expansion, canonical mappings, address masquerading, and other address mappings after the content filter. The `main.cf` setting of `receive_override_options` disables these mappings before the content filter.
- The `smtpd_xxx_restrictions` and `mynetworks=127.0.0.0/8` override `main.cf` settings. They turn off junk mail controls that would only waste time here.
- With `smtpd_authorized_xforward_hosts=127.0.0.0/8`, the scan transport will try to forward the original client name and IP address to the after-filter smtpd process, so that filtered mail is logged with the real client name and IP address.

smtpd_proxy_filter

Like a `content_filter`, but you can reject **before** the mail is being queued. Sort-of like policy delegation, but you can still look at the mails' content.

Pros and cons of `smtpd_proxy_filter`

- cons
  - needs to fast enough to inspect the content "while-u-wait"
  - must understand & speak SMTP
- pros
  - you can reject content before it's queued

ClamSMTP is an SMTP filter that allows you to check for viruses using the ClamAV anti-virus software. It accepts SMTP connections and forwards the SMTP commands and responses to another SMTP server. The ’DATA’ email body is intercepted and scanned before forwarding.


```
# apt-get install clamsmtp
... install ...
# usermod -g clamsmtp -G clamav clamsmtp
(get clamsmtp rights to access the clamav socket)
```

```
# ==========================================================================
# service       type  private unpriv  chroot  wakeup  maxproc command
#                     (yes)   (yes)   (yes)   (never) (100)
# ==========================================================================
smtp      inet  n       -       -       -       -       smtpd
   -o smtpd_proxy_filter=127.0.0.1:10026

localhost:10025
          inet  n       -       -       -       -       smtpd
   -o content_filter=
   -o smtpd_proxy_filter=
   -o receive_override_options=no_unknown_recipient_checks,
      no_header_body_checks
   -o smtpd_helo_restrictions=
   -o smtpd_client_restrictions=
   -o smtpd_sender_restrictions=
   -o smtpd_recipient_restrictions=permit_mynetworks,reject
   -o mynetworks=127.0.0.1
   -o smtpd_authorized_xforward_hosts=127.0.0.0/8
```


The log:

```
15:24:28 postfix/smtpd[11588]: connect from
         mail-ausfall.charite.de[193.175.70.131]
15:24:34 postfix/policyd-weight[18042]: decided action=\
         PREPEND X-policyd-weight: using cached result; rate: -7.6
15:24:34 clamsmtpd: 100000: accepted connection from: 127.0.0.1
15:24:34 postfix/smtpd[11591]: connect from localhost[127.0.0.1]
15:24:34 postfix/smtpd[11588]: NOQUEUE: \
         client=mail-ausfall.charite.de[193.175.70.131]
15:24:34 postfix/smtpd[11591]: 988A0B4617: \
         client=mail-ausfall.charite.de[193.175.70.131]
15:24:40 postfix/smtpd[11588]: disconnect from \
         mail-ausfall.charite.de[193.175.70.131]
15:24:40 clamsmtpd: 100000: from=Ralf.Hildebrandt@charite.de, \
         to=webmaster@arschkrebs.de, status=\
 VIRUS:GZip.ExceededFileSize
15:24:40 postfix/smtpd[11591]: disconnect from localhost[127.0.0.1]
```

And the bounce:

```
<webmaster@arschkrebs.de>: host mail.arschkrebs.de[88.198.105.204]
said: 550 Virus Detected; Content Rejected (in reply to end of DATA
command)
```


SMTP milter

This is the most recent addition to Postfix. That way you can add the buggyness of Sendmail to Postfix. No really, every milter I touched so far has been crap.

Postfix version 2.3 introduces support for the Sendmail version 8 Milter (mail filter) protocol. This protocol is used by applications that run outside the MTA to inspect SMTP events (CONNECT, DISCONNECT), SMTP commands (HELO, MAIL FROM, etc.) as well as mail content. All this happens before mail is queued.

Postfix Anti-Spam Workshop
└Milter

The reason for adding Milter support to Postfix is that there exists a large collection of applications, not only to block unwanted mail, but also to verify authenticity (examples: Domain keys identified mail, SenderID+SPF and Domain keys) or to digitally sign mail (examples: Domain keys identified mail, Domain keys). Having yet another Postfix-specific version of all that software is a poor use of human and system resources.
`http://sourceforge.net/projects/dkim-milter/`
`http://sourceforge.net/projects/sid-milter/`
`http://sourceforge.net/projects/dk-milter/`

Postfix version 2.4 implements all the requirements of Sendmail version 8 Milter protocols up to version 4, including message body replacement (body replacement is not available with Postfix version 2.3).

- pros
  - you can use software written for Sendmail
- cons
  - doesn't scale (pre-queue filtering)
  - bugs in either `libmilter`, the milter application or Postfix

Prerequisites

Milter applications have been written in C, JAVA and Perl, but this document deals with C applications only. For these, you need an object library that implements the Sendmail 8 Milter protocol. Postfix currently does not provide such a library, but Sendmail does.

DKIM-Milter

The goals of DKIM and DomainKeys are:

- assurance of sender identities
- protection against message tampering

DKIM standard: RFC 4871

The standard states the following, which applies to its predecessor DomainKeys (historical: RFC 4870) as well:

> *DomainKeys Identified Mail (DKIM) defines a mechanism by which email messages can be cryptographically signed, permitting a signing domain to claim responsibility for the introduction of a message into the mail stream. Message recipients can verify the signature by querying the signer's domain directly to retrieve the appropriate public key, and thereby confirm that the message was attested to by a party in possession of the private key for the signing domain.*

Signing to protect domains from phishing

The table shows average SpamAssassin score, grouped by a From address in a mail header, separately for mail **with** a verified first-party signature, and all the rest:

from_addr	avg score
@gmail.com	7.61
@yahoo.com	30.76
@ebay.com	33.33

from_addr	avg score
@gmail.com	-4.34
@yahoo.com	-1.26
@ebay.com	-6.54

Installation

In Debian/Ubuntu:

```
libmilter-dev - Sendmail Mail Filter API (Milter)
libmilter1 - Sendmail Mail Filter API (Milter)
dkim-filter - DomainKeys Identified Mail for Sendmail
```


Installation - `master.cf`

```
smtp      inet  n       -       -       -        -      smtpd
   -o milter_default_action=accept
   -o milter_macro_daemon_name=MTA
   -o smtpd_milters=inet:127.0.0.1:4443

localhost:10026
          inet  n       -       -       -        -      smtpd
   -o smtpd_client_restrictions=
   -o smtpd_helo_restrictions=
   -o smtpd_sender_restrictions=
   -o smtpd_recipient_restrictions=permit_mynetworks,reject
   -o mynetworks=127.0.0.0/8
   -o receive_override_options=no_unknown_recipient_checks
   -o content_filter=
   -o milter_default_action=accept
   -o milter_macro_daemon_name=ORIGINATING
   -o smtpd_milters=inet:127.0.0.1:4445
```

So, I do have one milter for incoming mail:
`smtpd_milters=inet:127.0.0.1:4443`
And one for outgoing mail:
`smtpd_milters=inet:127.0.0.1:4445`
This means you have to split incoming and outgoing traffic.

```
dkim-filter -f -u dkim-milter -b v -l -p inet:4443@127.0.0.1
```

is the one for incoming mail, since I'm running it from `runit`, it needs to stay in the foreground (`-f`), the user it runs as is `dkim-milter`, `-b v` indicates operational mode `verify`, `-l` uses `syslog()`, `-p inet:4443@127.0.0.1` specifies the socket to listen to.

Postfix Anti-Spam Workshop
Milter
setup

```
dkim-filter -u dkim-milter -b s -m ORIGINATING -p
inet:4445@127.0.0.1 -o X-MimeOLE Content-Class X-MS-Has-Attach
X-MS-TNEF-Correlator Thread-Topic Thread-Index X-OriginalArrivalTime
Disposition-Notification-To Received MIME-Version Content-Type
Content-Disposition Content-Transfer-Encoding User-Agent
-x /etc/dkim-filter.conf
```

is the one for outgoing mail. `-b s` indicates `signing`-mode, `-m ORIGINATING` is the MTA name (sendmail specific), and the list behind `-o` specifies headers which should be omitted when generating signatures.

Testing

Send mail to `dkim-test@testing.dkim.org`
Did that, got a crappy HTML mail saying:

```
DKIM Message Reflector Results
Authentication Results
testing.dkim.org; header.From=Ralf.Hildebrandt@charite.de; \
dkim=pass ( sig from charite.de/default verified; );

DKIM Processing Output
sig from charite.de/default verified with 0 appended bytes
End of Message
```


Testing II

And that message has a DKIM signature as well, from the headers:

```
DKIM-Signature: v=0.5; a=rsa-sha256; q=dns/txt; l=3506; t=1189951281;
c=relaxed/simple; s=dkimtest;
h=To:Content-Type:From:Subject:Content-Transfer-Encoding:MIME-Version;
d=dkim.org; i=mail@dkim.org;
z=From:=20mail@dkim.org
|Subject:=20DKIM=20reflector=20results
|Sender:=20
|To:=20Ralf.Hildebrandt@charite.de;
bh=MxWbMBjDcevsK2PvimESBDWGFz6ygDls1zB0tKTiXRw=;
b=kLM0+H1HFktgcIUc0A0wGXoN/fBmfWlIXM8UAwK/71gN+t191/WI2w/N5nj/57LtbRZA
1Br49ITIpjmnuxjhTpk2tFj+e2ikh+dZzEfIP8aNsAkriikHk7R0vkpuSgeKtn2iEjBOZx
6WSxkGOyCdVo7xjjbaS7mtEhA=;
Authentication-Results: testing.dkim.org; header.From=mail@dkim.org; \
        dkim=pass ( sig from dkim.org/dkimtest verified; );
```

So it seems to work both ways.

Questions?

The slides are available at:

`http://www.arschkrebs.de/slides/lisa2007-antispam-slides.pdf`