gmilt manual

Release 1.16

Note: See “Release Notes” on page 23.

Contents

1. Overview

The name, *gmilt*, stands for “Graylist Milter.” It is a persistent daemon can run in the background on Unix based machines. The *gmilt* milter was designed with the following high-level requirements in mind:

- The daemon must continue to run despite network or system failures.
- The daemon must produce logs or records minimally of errors encountered.
- The daemon must be secure and must not allow intrusion to the hosting machine.
- The daemon must contain hooks for debugging and diagnosing problems.

With these high level requirements in mind, the *gmilt* was written utilizing the following software requirements:

The daemon must be written in C for best performance.

- The daemon must support the standard Milter API protocol.
- The daemon must be completely configurable with a configuration file.

The *gmilt* milter is built using *autoconf* so that it may be easily ported to new and unforeseen operating systems and version of operating systems.

The *gmilt* milter is entirely configuration file driven. One way to run it by hand might look like the following:

```
./gmilt -c gmilt.conf
```

This will launch the *gmilt* milter program as a constantly running program until killed. Because the *gmilt* milter is configuration-file-centric, we describe that configuration file first and follow that description by describing the other aspects of the program.

2. The gmilt.conf file

The *gmilt* daemon employs a single configuration file. The file is typically called, *gmlt.conf*. The *gmilt.conf* file provides most of the information needed for the *gmilt* milter to run and function properly on your machine and inside your environment. Empty lines in the configuration file and lines that begin with a # character are ignored. Each line may only specify a single configuration item in the format:

```
name = value
```

There may be arbitrary white space surrounding the name, the =, or the value. Quotation marks have no special meaning. Some items (like **listen**=) may appear only once, while other (such as **greylist**=) may appear multiple times. Note that unknown configuration items are silently ignored.

2.1. Configuration: listen=

The **listen=** specifies, in standard Milter format, how to listen for connections from the MTA.

```
listen = inet:8090@192.168.0.8
```

The **listen=** is mandatory and must be listed in the configuration file. If the value specified is incorrect for a Milter to start, it will result in the following error being logged, as for example:

```
gmilt: Unable to create listening socket on conn inet:25@192.168.0.8
Unable to bind to port inet:25@192.168.0.8: Permission denied.
```

This configuration item may only appear once, and its value should match the value you gave to your MTA.

2.2. Configuration: runasuser=

The *gmilt* milter will refuse to run if run by *root*. If you need to start it as *root*, you will need to specify this **runasuser** configuration option:

- **runasuser=**login name — Means become that login user.
- **runasuser=**login ID — Means become the user associated with that UID

If you attempt to run as *root* without setting this **runasuser** configuration option, the following error will print and be sysloged:

```
Attempt to run as root forbidden
```

If the user specified by this **runasuser** configuration option does not exist (e.g. “foo”) the following error will print and be sysloged:

```
runasuser foo: No such user.
```

An attempt to run as a different users that fails, will cause the program to refuse to run. So you should only set this **runasuser** configuration option when you plan to run this milter by *root*.

2.3. Configuration: pidfile=

Some /etc/init.d startup scripts require that the program create a process-id file in order for the startup script to later stop the program. If your startup script needs that information, you can cause *gmilt* to create it using this configuration item:

```
pidfile = /var/run/gmilt.pid
```

But note there are a few restrictions for using this **pidfile=** configuration item:

- If gmilt is run by or as *root*, the **pidfile=** value must begin with “/var/run”
- If gmilt is run by or as *root*, the **pidfile=** value must not contain “..”

If *gmilt* is run as an ordinary user, there is no restriction on the name of the PID file. If it cannot be written, an error will be logged, but *gmilt* will continue to run. Also at shutdown, if the PID file cannot be removed, an error will be logged but *gmilt* will exit anyway.

2.4. Configuration: rejectmode=

If not told to do otherwise, anything found wrong with an email message will cause it to be rejected (bounced). Beginning with V1.10 of *gmilt*, you may chose from among other ways to handle bad messages:

- **rejectmode = reject** means to do the default behavior of rejecting bad messages.
- **rejectmode = defer** means to accept and silently discard bad messages.
- **rejectmode = quarantine** means to accept the massage and quarantine it in the queue.
- **rejectmode = tempfail** means to reject by tempfailing the message back to the sender.
- **rejectmode = annotate** means to accept and deliver the message but to add an “X-Gmilt: Reject” header to the message.

How you choose to handle bad messages is purely a matter of your site’s policy.

2.5. Configuration: greylist=

The **greylist=** specifies the parts of an email message that will be used to identify when it is re-delivered. Here each part is specified on a line by itself each using this **greylist=** item. The legal parts are:

- **greylist=ipnum** — The first 32 characters of the connecting host's IP address
- **greylist=rcpto** — The first 128 characters of the first RCPT TO: envelope value
- **greylist=mail** — The first 128 characters of the first MAIL FROM: envelope value
- **greylist=subject**— The first 128 characters of the first Subject: header
- **greylist=msgid** — The first 128 characters of the first Message-Id: header
- grey**list=from** — The first 128 characters of the first From: header
- **greylist=to** — The first 128 characters of the first To: header
- **greylist=sender** — The first 128 characters of the first Sender: header

So, for example, to check for the retry using the Subject: and Message-ID: headers in that order, simply specify:

```
greylist = subject
greylist = msgid
```

Note that if you change these definitions and reload the configuration file using a SIGHUP signal, any previously temp-failed messages will be temp-failed a second time as the new values are used.

2.6. Configuration: whitelist=

The *gmilt* daemon accept without further processing any sending IP address listed as a whitelisted IP address.

```
whitelist = 127.0.0.1
```

Note that there must be a single address per **whitelist=** expression (use **whitelist_cidr=** to list a range of addresses). Comparisons are text based as left-side matches. That is, for example:

```
whitelist = 192.168.0
```

This will match any address in the subnet 192.168.0. Care should be observed using this method because **whitelist=** defined IP addresses will become an exact match beginning with the next *gmilt* release 1.10.

2.7. Configuration: whitelist_cidr=

The *gmilt* daemon accept without further processing any sending IP address listed as a whitelisted IP address. A range of IP addresses can be listed using CIDR notation with the **whitelist_cidr=** expression:

```
whitelist_cidr = 192.168.0.0/24
```

Note that there must be a full IP-address, followed by a slash, then an integer expression of bits for each **whitelist_cidr=** expression (the **whitelist=** expression can be used to list single IP address). The **-t** command-line switch (See §3.3. on page 19) can be used to test an expression, for example:

```
./gmilt -t cidr:123.45.67.8/29
123.45.67.8
123.45.67.9
123.45.67.10
123.45.67.11
123.45.67.12
123.45.67.13
123.45.67.14
123.45.67.15
```

This switch allows you to see how a CIDR expression will be expanded before you put it into your *gmilt.conf* file.

2.8. Configuration: blacklist=

The *gmilt* daemon can blacklist (unconditionally reject) based on the sending IP address when listed as a blacklisted IP address.

```
blacklist = 123.45.67.89
```

Note that there must be a single address per **blacklist=** expression. Comparisons are text based and must match exactly to reject. That is, for example:

```
blacklist = 192.168.0.5
```

This will match the address 192.168.0.5, but will not match 192.168.0.55. Any blacklisted address will be rejected at the connection stage, before any envelope information is received. But note that a whitelisted address can never be blacklisted because *gmilt* accepts before rejecting.

2.9. Configuration: blacklist_cidr=

The *gmilt* daemon can blacklist (unconditionally reject) based on the sending IP address when that address is listed as a blacklisted IP address. A range of IP addresses can be listed using CIDR notation with the **blacklist_cidr=** expression:

```
blacklist_cidr = 109.167.192.225/29
```

Note that there must be a full IP-address, followed by a slash, then an integer expression of bits for each **blacklist_cidr=** expression (the **blacklist=** expression can be used to list single IP address). The **-t** command-line switch (See §3.3. on page 19) can be used to test an expression, for example:

```
./gmilt -t cidr:109.167.192.225/29
109.167.192.224
109.167.192.225
109.167.192.226
109.167.192.227
109.167.192.228
109.167.192.229
109.167.192.230
109.167.192.231
```

This switch allows you to see how a CIDR expression will be expanded before you put it into your *gmilt.conf* file.

2.10. Configuration: blacklistauto=

When the envelope sender is rejected, the IP address of the rejected sending site can be used to blacklist future connections. Generally envelope senders are rejected to prevent address harvesting by bots. Because such address harvesting presents a continuously high load on an MTA it would be better to reject the connections before the MAIL FROM: was sent.

This **blacklistauto=** expression lists one of three possible ways to handle those bad IP addresses. The choices are:

- **blacklistauto=file** — Means to append those bad addresses to a file named “blacklist” in the current directory. Each line of that file will look like this:
  `blacklist = 12.34.56.78`
  The contents of that file could be added to your *gmilt.conf* file periodically by hand.
- **blacklistauto=auto** — Means to automatically add that address to the internal hash table of blacklisted IP addressees. This allows automatic rejection, but the new addresses will be lost the next time the *gmilt.conf* file is reloaded. This **auto** setting could be useful if you want immediate relief but don’t want to permanently block those addresses.

- **blacklistauto=both** — Means to both add to a file and to update the internal hash table.

2.11. Configurable Header Screening

You may screen the values of particular headers for strings to reject. The name of the header is on the left, followed by an equal sign, and then the string to reject. For example:

```
from = Rolex     ← Reject From: headers containing “Rolex”
```

The only header that can be screen in this manner are:

- **from =** ← Reject string inside **From:** header
- **to =** ← Reject string inside **To:** header
- **reply-to =** ← Reject string inside **Reply-To:** header
- **sender =** ← Reject string inside **Sender:** header
- **subject =** ← Reject string inside **Subject:** header

The string specified is a literal string, not a regular expression. All comparisons are case-insensitive, that is, “BOB” will match “Bob” and “bob”. The comparison is also a sub-string match, so “bob” will match “Bobbin” and will match “bobolink” but will not match “Bo Bo The Clown.” The reason the last string did not match is because internal whitespace is preserved. For example:

```
subject = Bo The Clown
```

Here, the string will match “Bo Bo The Clown” and “bo bo the clown” as expected, but will also match “Elbo bo the clown.” Clearly specification of strings must be done with care.

If a string in a header’s value is a match, the message is rejected and the following is logged, as for example:

```
ip=190.233.117.112, REJECT=Word “vagary” found in From: “Viagra Sample”
<tinctureyz9729@example.com>
```

Note that the entire header’s value is searched for a match, so the following will also reject the above header:

```
from = @example.com
```

You may list any header specification multiple times. The comparisons will be made top down in the order listed in your configuration file.

2.11.1. Regular Expression Support

Beginning with V1.7 of the gmilt milter, a regular expression can be used with these configuration items. To use a regular expression, merely prefix the string with the word regex and a colon, as for example:

```
from = regex:\<viagra\>
```

Everything following the colon to then end of the line is taken as the regular expression. Note that regular expressions are only compiled at run-time, so will not be reported as errors in the configuration file. If a regular expression fails to compile, it will be logged with an error like the following:

```
“regex:[A-z]123Rubber” regexec error: Invalid endpoint in range expression
```

Regular expression are always compiled to make case-insensitive comparisons, and to support Extended Regular Expressions. Each expression is only evaluated once, so expressions requiring multiple steps will not be supported.

2.12. Configurable Envelope Screening

Beginning with V1.8 of the gmilt milter, the envelope sender and envelop recipients can be screened and rejected. The name of the envelope field is on the left, followed by an equal sign, and then the string to reject. For example:

```
mailfrom = ViagraMareting@    ← Reject MAIL From: containing this string
rcptto   = Bob@olddomain.com  ← Reject RCPT To: containing this string
```

The string specified is a literal string, not a regular expression. All comparisons are case-insensitive, that is, “BOB” will match “Bob” and “bob”.

The comparison is also a sub-string match, so “bob” will match “Bobbin” and will match “bobolink” but will not match “Bo Bo The Clown.” The reason the last string did not match is because internal whitespace is preserved.

If a string in a envelope’s value is a match, that sender or recipient is rejected and the following is logged, as for example:

```
ip=190.233.117.112, REJECT=Found ViagraSales@ in <ViagraSales@example.com>
```

You may list any envelope specification multiple times. The comparisons will be made top down in the order listed in your configuration file.

2.13. Configure: badtofrom=

The *gmilt* milter can reject messages that are received from off-site, but where at least one To: header matches exactly the address in at least one From: header, and where the address is in the local domain. For example, consider that your site is example.com and you receive an email message with the following two headers:

```
From: bob@example.com
To: bob@example.com
```

If this message arrives on the external facing interface, it can be rejected by specifying:

```
badtofrom = true
```

Any true value (True, true, Yes, yes) turn on this check. Omitting this configuration item, or setting it to false (False, false, No, no) turns off this check.

2.14. Configure: badhihead=

The *gmilt* milter can reject messages that contain header lines with binary data embedded but not encoded. By setting the following option to true (True, true, Yes, yes) you cause the *gmilt* milter to reject any message that has characters with the high-bit set in any header's value.

```
badhihead   = true
```

Omitting this configuration item, or setting it to false (False, false, No, no) turns off this check. If a message is rejected for this reason, the following is logged:

```
ip=111.67.201.186, REJECT=Binary data found in “From:” header
```

The header contents are not logged because of the risk of logging binary data. Instead the QID of the message is also logged so you can match this log line with the MTA's log messages.

2.15. Configure: noatinmid=

The *gmilt* milter can check the Message-Id: header to insure it is somewhat properly formed:

```
noatinmid     = true
```

This causes the *gmilt* milter to check the Message-Id: header's value for the presence of an @ character. If the Message-Id: header's value lacks that character the message is rejected with the following logged message:

```
ip=58.39.144.23, REJECT=Required “@” not found in “Message-ID:” header
```

Omitting this configuration item, or setting it to false (False, false, No, no) turns off this check.

2.16. Configure: rejectforged=

The *gmilt* milter can reject (bounce) mail that the MTA considers forged:

```
rejectforged     = true
```

This causes the *gmilt* milter to check the MTA's ${client_resolve} macro for the string “forged” and if found, to reject the message with the following logged error:

```
ip=58.39.144.23, REJECT=Sendmail marked host as forged
```

Omitting this configuration item, or setting it to false (False, false, No, no) turns off this check.

2.17. Configure: missing=

The *gmilt* milter can reject (bounce) mail that is missing a header that you consider required:

- **missing=from** ← Require at least one **From:** header
- **missing=to** ← Require at least one **To:** header
- **missing=msgid** ← Require at least one **Message-Id:** header
- **missing=subject** ← Require at least one **Subject:** header

Omitting this configuration item turns off enforcement of this policy. You may list one or more headers, one per line, as for example the following requires a **To:** and a **From:** header to be in the message:

```
missing = from
missing = to
```

There is currently no way to require any arbitrary header.

2.18. Statistics

It is now possible to follow the behavior of this milter without the need to parse logs. If configured to do so, *gmilt* will launch a thread to listen for network connections. This listening interface can be run in any of three modes:

- HTML allows a connection from a web browser to view the statistics in html format.
- TEXT allows a command-line program (such as telnet) to connect and view the statistics in formatted plain-text.
- XML allows any program to connect and be fed the statistics in xml format, but without a style sheet.

Only one mode can be used at any given time, and the mode is set in the configuration file:

2.18.1. Configure: stats_how=

The **stats_how=** configuration item specifies in what of three forms of output the statistics interface should present its output. The choices are:

- **stats_how= html** ← Output statistics in HTML format
- **stats_how= text** ← Output statistics in TEXT format
- **stats_how= xml** ← Output statistics in XML format

The how is case-insensitive, which means HTML is the same as html. If anything other than these three choices is made the default becomes HTML.

2.18.2. Configure: stats_interface=

The listen statistics interface is an IP address. It may either be a host or a network address. Some addresses are special:

- **stats_interface= 127.0.0.1** ← The loopback interface
- **stats_interface= INADDR_ANY** ← All physical interfaces
- **stats_interface= 0.0.0.0** ← All physical and all virtual interfaces

In general is is best to restrict queries to the loopback interface unless you are behind a router that prevents outside connections to arbitrary ports on your machine. If you are, for example, on a non-routable network (such as 192.168. or 10.) you might be safe to listen on all interfaces.

Note that this **stats_interface=** and the **stats_port=** must be specified for the statistics interface to function.

2.18.3. Configure: stats_port=

The port upon which the statistics interface will listen. This should be a non-privileged port (below 1024 on many systems, below 4096 on others). If it is too low, the gmilt will not be able to listen and will syslog an error. If it is a non-privileged port but one already used by another program, the *gmilt* will not be able to listen and will syslog an error.

```
stats_port= 8765
```

Note that this **stats_port=** and the **stats_interface=** (previous section) must be specified for the statistics interface to function.

2.18.4. Configure: stats_cache=

If you ever need to stop and restart the *gmilt* milter, all your statistics will be lost because they are stored in memory. If you wish your stats to be continuous across stops and restarts, you must define this **stats_cache=** configuration item:

```
stats_cache= /etc/mail/milter/gmilt/stats.cache
```

If the file cannot be written when as the *gmilt* milter is shutdown, *gmilt* will syslog an error and shutdown anyway. When it starts up again, it will try to read that file. If the file does not exist, the gmilt will silently skip loading its contents.

The **stats_cache=** file has been designed to be forward compatible. New items will be added to the end and will not throw off the counts of earlier versions.

With the beginning of version 2 stats, as of gmilt 1.10, reading a version 1 cache file into a version 2 stats gmilt will cause the old stats to be discarded. This is done because the old stats had no time component that could be used to generate rates. As of version 2 stats, a time component has been added so that rates can be computed. All version 2 rates are per-day.

2.18.5. Configure: stats_reset=

Beginning with Gmilt 1.10, you may decide to have every read of stats zeroed (reset) immediately after each stats read. You set this behavior by defining:

```
stats_reset = true
```

If the **stats_reset=** configuration option is set to “true” or “yes” (case insensitive) stats will be reset after each and every read. If the **stats_reset=** configuration option has any other value or is omitted, stats are preserved across reads.

2.18.6. Sample HTML output:

Note that some lines may not appear if your MTA is older than some required features. Also some items may not appear unless they are configured.

Table 1: Example of HTML Output

Gmilt 1.12, Stats V2, Day 136, Hour 3281

WHAT	COUNT	PER/DAY	PER/HOUR
Milter Connections	3278	819	46
Milter Memory Failures	0	0	0
Accept Whitelist	200	50	0

Table 1: Example of HTML Output

Gmilt 1.12, Stats V2, Day 136, Hour 3281

Defer Greylist	892	233	1
Accept After Graylist	10	2	0
Reject Blacklist	8	2	0
Reject Bad FROM Value	21	5	1
Reject Bad To: Header Value	0	0	0
Reject Bad Subject: Header Value	0	0	0
Reject To == From	0	0	0
Reject Header With Hi-Bit Set	2	0	0
Reject MTA Said Forged	0	0	0
Reject MESSAGE-ID Lacked @	0	0	0
Reject To: Header Missing	15	3	0
Reject From: Header Missing	0	0	0
Reject Message-Id: Header Missing	0	0	0
Reject Subject: Header Missing	0	0	0
Reject Bad MAIL FROM: Envelope Value	0	0	0
Reject Bad RCPT To: Envelope Value	0	0	0
Count Bad/Unknown Recipients	1537	384	6
Harvested Connect Onlys	2663	8	2
Harvested IP Rejected	1200	0	0
Harvest IP Blacklisted	39	0	0

2.19. Harvesting Bots Detected

Beginning with release 1.12, gmilt automatically detects two different types of bot attacks:

- Connections made to see only if an MTA is listening on port 25.
- Connections made to discover email addresses.

2.19.1. Harvest Listening MTAs

When a bot connects just to see if an MTA is listening, sendmail will report that the connection ended and the connecting client issued no MAIL/EXPN/VRFY/ETRN. Likewise, the milter can detect the same thing, and when it does it logs a message like the following:

```
May 21 09:08:05 example gmilt-V1.12: [ID 527766 mail.info] glibmilter.c(622):
p4LG80w4023258: harvest 92.21.198.26: No MAIL/EXPN/VRFY/ETRN
```

The *gmilt* milter then adds this IP address to an internal list of harvested IP addresses. Thereafter, if the Bot connects to harvest again, the connection will be rejected. But note that a harvested IP address will only be rejected for an hour, then allowed again.

```
May 21 09:23:05 example gmilt-V1.12: [ID 527766 mail.info] glibmilter.c(622):
p4LG80w4023258: harvest 92.21.198.26: REJECT=Harvested Address
```

If an address, within that one hour reject window, tries the same attack two more times, that IP address is moved out of the harvest list and added as a blacklisted address. The internal harvest list and internal blacklist are transient and will be discarded whenever *gmilt* restarts.

2.19.2. Harvest Recipient Addresses

When a bot connects to see if a recipient address is good, but the address is bad, the Bot will disconnect immediately after the RCPT TO: envelope command. The gmilt milter counts the number of good recipients and if there were zero good recipient it logs the following:

```
May 21 09:08:05 example gmilt-V1.12: [ID 527766 mail.info] glibmilter.c(622):
p4LG80w4023258: harvest 92.21.198.26: Zero recipients
```

The gmilt milter then adds this IP address to an internal list of harvested IP addresses. Thereafter, if the Bot connects to harvest recipients again, the connection will be rejected. But note that a harvested IP address will only be rejected for an hour, then allowed again.

May 21 09:23:05 example gmilt-V1.12: [ID 527766 mail.info] glibmilter.c(622): p4LG80w4023258: harvest 92.21.198.26: REJECT=Harvested Address

If an address, within that one hour reject window, tries the same attack two more times, then the IP address is moved out of the harvest list and added as a blacklisted address. The internal harvest list and internal blacklist are transient and will be discarded whenever *gmilt* restarts.

2.19.3. Harvesting always on

Harvest detection is always turned on. It cannot be turned of using a configuration option. If there is a request for such an option, one may be added in a future release.

2.20. SPF (Sender Policy Framework) Checks

Release 1.13 of the *gmilt* milter now can perform SPF checks based on the IP address of the connecting host, the envelope **MAIL From:** address, and (if that address is empty or is the literal expression "MAILER-DAEMON") the HELO/EHLO domain. There are six configuration options that tune how the gmilt milter will handle SPF result:

```
####### HANDLE SPF RECORDS ################################################
# SPF_Policy can be: off, monitor, quarantine, reject (on FAIL, default off)
# SPF_On_None if true apply policy to a missing SPF record (default false)
# SPF_On_Soft if true apply policy to an SPF soft fail (default false)
# SPF_Header if true will add an Received-SPF: header (default false)
spf_policy = monitor
spf_on_none = false
spf_on_soft = false
spf_header = true
spf_pass_no_greylist    = false
```

The logic used by the gmilt SPF checker is as followed:

- Look up the TXT record for the envelope sender domain.
- If the envelope sender lacks a domain, look up the TXT record for the HELO/EHLO domain.
- If no TXT record is found, repeat the check for the T_SPF type (99).
- If the query times out and we found a CNAME, re-query the CNAME.
- Accept any record returned that contains v=spf1.
- Accept any record returned that begins with spf2.0.
- Parse the record performing DNS lookups as required.
- Any SPF record causing more than 10 DNS lookups is an automatic FAIL.

2.20.1. The spf_policy Configuration Option

The **spf_policy** configuration option specifies how you want to use SPF records. There are four possible values you may proved for this option:

- **off** — Means that you will not be checking SPF records, nor will you produce an SPF-Results: header.
- **monitor** — Means that you will not apply any policy to SPF records, but will record the outcome of each check in an SPF-Results: header and an Authentication-Results: header.
- **quarantine** — Means that any messages that FAIL the SPF check will be quarantined.
- **reject** — Means that any messages that FAIL the SPF check will be rejected.

Note, even if you reject messages that FAIL the SPF check, that does not mean you will stop spam and phishing messages. Bad and hostile mail often properly supports good SPF records.

2.20.2. The spf_on_none Configuration Option

Normally a missing SPF record is not considered an SPF failure. If, however you set this option to true, you enable the gmilt milter to interpret a missing SPF record as a FAIL.

- **spf_on_none = true** — Means to interpret a missing SPF record as a FAIL. If your **spf_policy** is set to monitor, the FAIL will be recorded. If **spf_policy** is set to quarantine, the FAIL will cause the message to be quarantined. And if **spf_policy** is set to reject, the FAIL will cause the message to be rejected.
- **spf_on_none = false** — Means to allow a missing SPF record to be a SOFT FAILURE and allowed (unless **spf_on_soft** below is true).

2.20.3. The spf_on_soft Configuration Option

Normally a SOFT FAILURE is caused because no SPF record was found or because the check failed but a ~all or a +all allowed the failure to otherwise succeed. This **spf_on_soft** configuration file option, if set to true, causes a SOFT FAILURE to be interpreted as a FAIL.

- **spf_on_soft = true** — Means to interpret a SOFT FAILURE as a FAIL. If your **spf_policy** is set to monitor, the FAIL will be recorded. If **spf_policy** is set to quarantine, the FAIL will cause the message to be quarantined. And if **spf_policy** is set to reject, the FAIL will cause the message to be rejected.
- **spf_on_soft = false** — Means to allow a SOFT FAILURE to succeed.

2.20.4. The spf_header Configuration Option

The spf_header configuration option determines whether or not the **SPF-Results:** header will be added to the inbound message.

- **spf_header = true** — Always insert this header
- **spf_header = false** — Never insert this header

2.20.5. The spf_pass_no_greylist Configuration Option

Normally any inbound mail that is not whitelisted, will be either rejected or grey-listed. The **spf_pass_no_greylist** configuration option causes any mail that passes the SPF check to be delivered without first being grey-listed.

- **spf_pass_no_greylist=true** — Whitelist addresses that pass the SPF check
- **spf_pass_no_greylist=false** — Graylist addresses that pass the SPF check

Setting this option to true is not recommended, however, because spam and phishing can pass SPF checks, yet still fail graylisting.

3. Command-Line Arguments

The *gmilt* milter is a program that is started using the command-line. That command line can either be executed by hand or can be executed by a startup script to run unattended. If run with no arguments the following is printed:

```
Usage: gmilt [-t test] [-V] -c configfile
        -t cidr:address/bits
        -t spf:helodomain,mailfromaddr:IPv4
```

Note that only the -c switch is mandatory. All the others are optional.

3.1. The -c switch

The -c switch tells the *gmilt* milter the location of its configuration file. The milter reads and parses its configuration and tries to start running. Errors in the configuration file may prevent it from running if first started. If the configuration file is automatically re-loaded at run-time, errors are logged but the milter continues to run.

3.2. The -D switch

The **-D** command-line switch has been removed.

3.3. The -t Testing Switch

The **-t** command-line switch provides a general purpose testing tool. When you use **-t** you may omit the **-c**. Recall that running gmilt with no arguments produces a usage statement:

```
%./gmilt
Usage: gmilt [-t test] [-V] -c configfile
        -t cidr:address/bits
        -t spf:mailfromaddr:helodomain:IPv4
```

3.3.1. The -t cidr: test

The -t cidr: test is used to find out how the *gmilt* program will parse CIDR notation addresses. For example:

```
./gmilt -t cidr:123.45.67.8/30
```

```
123.45.67.8
123.45.67.9
123.45.67.10
123.45.67.11
```

3.3.2. The -t spf: test

The **-t spf:** test is used to see how the milter handles SPF records. For example:

```
%./gmilt -t spf:bounce.gmail.com:bob@gmail.com:123.45.67.8
        gmail.com: v=spf1 redirect=_spf.google.com
        _spf.google.com: v=spf1 ip4:216.239.32.0/19 ip4:64.233.160.0/19
ip4:66.249.80.0/20 ip4:72.14.192.0/18 ip4:209.85.128.0/17 ip4:66.102.0.0/20
ip4:74.125.0.0/16 ip4:64.18.0.0/20 ip4:207.126.144.0/20 ip4:173.194.0.0/16 ?all
        _spf.google.com: ?all, status= 123.45.67.8 Not found, but: NEUTRAL
Received-SPF: pass mailfrom=gmail.com; client_ip=123.45.67.8; helo_domain=
bounce.gmail.com
```

But note that this test omits the SPF configuration (See “SPF (Sender Policy Framework) Checks” on page 17) file settings so, for example, a soft failure (as here) might be configured to be a hard failure in your configuration file.

This command-line test can be used to diagnose SPF problems at other domains, and possibly your own. In addition to the normal SPF-style reporting, this **-t spf:** switch will also diagnose problems such as:

- Illegal or bad us of SPF macros
- Badly formed expressions such as redirect: without a domain specification
- Illegal data, such as ip4 followed by a non-address expression

But note that Spf2.0 (Sender Identity) are not specially syntax checked. So, for example, and error such as Spf2.0/mfrompra will be ignored, and the data following it will be used despite that error.

4. How to Install

For this description we presume you already downloaded the distribution:
`gmilt-1.16.tar.gz`

Unpack like this:

```
% tar xzvf gmilt-1.16.tar.gz            ← Linux
% gzcat gmilt-1.16.tar.gz | tar xvf -   ← Solaris
```

Change into the distribution directory
`gmilt-1.16`

There you run the following two commands to build the daemon:

```
./configure
make -s
```

To install the daemon run:

```
make install
```

Be certain to set up a script in /etc/init.d to start the daemon when your machine reboots. Because such scripts vary so widely between flavors of Linux and between Solaris and OS-X, we leave the creation of such a script to you.

You may use the run.sh script in the source directory as one way to insure that the daemon runs even if it faults and fails.

5. Known Bugs

A few problems are currently not detected, so we treat these areas as potential bugs.

5.1. Bug: No fork()

The daemon currently does not have a way to fork() and detach to place itself into the background. This will be fixed in a future release.

5.2. Bug: No Configuration Check

The validly of the configuration file is not checked at startup. A future release will fix this problem.

6. Release Notes

6.1. For Release 1.8

- A statistics interface has been added. Four new configuration file settings control it. See “Statistics” on page 12.
- The milter can now create and remove a PID file. See “Configuration: pidfile=” on page 5.
- Envelope recipient and sender can now be screened and used for rejecting. See “Configurable Envelope Screening” on page 10.
- Blacklisting of IP addresses is now supported. See “Configuration: blacklist=” on page 7.

6.2. For Release 1.9

- It was possible to reject a white-listed IP address for other reasons (such as also being blacklisted). Beginning with this release, no white-listed IP address can be rejected. See “Configuration: whitelist=” on page 6
- CIDR notation is now supported for whitelisted (See “Configuration: whitelist_cidr=” on page 7) and blacklisted (See “Configuration: blacklist_cidr=” on page 8) addresses. The next release will eliminate the left hand side matching of whitelist addresses.
- If an envelope sender or recipient is rejected, an attempt is made to set the SMTP error to 421. If that succeeds, the milter returns temp-failure causing the connection to be dropped by sendmail.
- A new configuration item “blacklistauto” (See “Configuration: blacklistauto=” on page 8) can cause envelope sender rejections to either be written to a “blacklist” file or auto-added to the internal hash table. This can be handy when fighting a bot attack trying to harvest email addresses.

6.3. For Release 1.12

- Stats are now version 2 which add the duration of stats so rates can be calculated (See descriptive paragraph on page 14).
- A new stats option **stats_reset=** allows the stats to be zeroed (reset to zero) after each and every time they are read (See “Configure: stats_reset=” on page 14).
- A new **rejectmode=** configuration item allows bad messages to be handled by other means than merely rejecting them (See “Configuration: rejectmode=” on page 5).
- Bots that harvest listening mta addresses and recipient names are now automatically detected and rejected (See “Harvesting Bots Detected” on page 15).

6.4. For Release 1.13

- SPF checking and policy have been added. See “SPF (Sender Policy Framework) Checks” on page 17.
- BUG: A whitelisted address could still end up on the soft BLACKLIST, but without harm because we accept before reject. This (logging only) defect has been fixed.
- The **-D** command-line switch has been retired. See “The -D switch” on page 19
- The **-t** command-line switch has been added. See “The -t Testing Switch” on page 19
- The **runasuser** configuration option has been added to prevent running as *root*. See “Configuration: runasuser=” on page 4

6.5. For Release 1.14 (November 21, 2012)

- Portability Issues fixed. Now builds cleanly on OS-X, Linux, FreeBSD, and Solaris
- No longer requires openssl to build.
- This milter can run in front of or behind a DKIM milter and when followed by the OpenDMARC milter will interoperate correctly.
- Fixed a bug that broke the XML and HTML stats displays.

6.6. For Release 1.15 (April 16, 2013)

- A randomly filled local variable fed to regcomp() could cause the program to fault. This has been fixed.
- Spam from open relays have user From: parts in Camel_Camel@ format. When discovered, an X-Gmilt-Camel: header is inserted and can be used to screen for possible spam.
- A few strlcpy(buf, ..., sizeof 256) expressions broke some SPF checks. These mistakes have been changed into strlcpy(buf, ..., sizeof buf)
- SPF ptr expressions are now correctly parsed.
- A double free() in gpolicy.c could fault on Linux. This mistaken free has been fixed.

6.7. For Release 1.16 (February 03, 2014)

- Fixed a bug that prevented a white-listed IP address from having an SPF check. This could cause valid sites to fail DMARC when white-listed.