

Harald Radi (harald.radi@nme.at) – <http://www.nme.at>
Felix Schwenk (felix.schwenk@nme.at) – <http://www.nme.at>

1 Filtering Mail with php-milter

Due to the increasing amount of spam and viruses spread via email the demand for mail filtering solutions, including spam rejection, virus filtering, and content control, is higher than ever. In this paper we will present a way of how you can write mail filters in PHP using Sendmail and the php-milter SAPI. This way you can match dispatcher addresses against your local blacklist, strip off attachments or identify mass spammers based on site statistics and much more, all within PHP scripts.

1.1 Introduction

Beside a recent version of PHP you need the Sendmail mailer daemon, which provides a mail filtering API called Milter allowing third-party software to process messages as they pass through the mail transfer system. We will introduce you to a PHP SAPI called php-milter that is currently only available via CVS from cvs.php.net and will be part of the next PHP release.

php-milter runs as a standalone daemon and takes use of Sendmail's mail filtering API to delegate the Milter callback functions to a PHP script. This way all processed mails are piped through an easily maintainable script and can be altered, rejected or discarded before they are fully delivered to the receivers inbox.

A description of the necessary components and a guidance how to configure them to use PHP as a mail filter are provided here. After that you will be introduced into the filter API and how it can be used in a PHP script.

1.2 Setup and Configuration

To use PHP, to filter mail, a correct installation and configuration of Sendmail and PHP is required. We will guide you through the complete setup and configuration process based on the GNU/Linux operating system. The information provided here can easily be adopted to set up a similar system on a different platform.

1.2.1 Building Sendmail with Milter support

After downloading and unpacking a recent version of Sendmail from <http://www.sendmail.org> you have to modify the site configuration to include the Milter API in the build process. Therefore you have to append the following lines to your dev-tools/Tools/site.config.m4 file:

Setup and Configuration

```
APPENDEDEF(`conf_sendmail_ENVDEF', `-DMILTER')dnl
APPENDEDEF(`conf_libmilter_ENVDEF', `-D_FFR_MILTER_ROOT_UNSAFE')dnl
```

Listing 1: site.config.m4

This file even might not exist, in that case simply create an empty file adding those two lines.

Done with that start the build process by running the following commands:

```
./Build
make install
```

Listing 2: Building Sendmail

If you don't experience any problems during the build you should be done with the installation of Sendmail and Milter.

1.2.2 Building php-milter

Currently php-milter is only available via CVS. The user for accessing PHP's CVS tree is cvsread and the password is phpfi.

```
cvs -d pserver:cvsread@cvs.php.net:/repository login
cvs -d pserver:cvsread@cvs.php.net:/repository co -r PHP_4_3 php4
cvs update -d php4/sapi/milter
```

Listing 3: Getting PHP from CVS

A more detailed explanation on how to build php out of CVS is available online at <http://www.php.net/anoncv.php>.

After having PHP checked out successfully you have to run the commands from listing 4 to configure and build it. Note that you also have to specify all the extra extension you want to use when running the configure script. More on that is available online at <http://www.php.net/install.configure>.

```
./buildconf
./configure --with-milter
make
make install
```

Listing 4: Building PHP from scratch

1.2.3 Configure Filters

Having everything in place now you have to make Sendmail aware that you want to process all mail with your PHP script. This is done by the `INPUT_MAIL_FILTER` macro in your `sendmail.mc` file, which usually resides in the `/etc/mail` folder of your system. Listing 5 shows a very basic configuration that registers three mail filters. A detailed explanation of all the macros used here and lots of other configuration scenarios can be found online at the Sendmail page (<http://www.sendmail.org>).

```
divert(-1)
include(`/usr/share/sendmail/cf/m4/cf.m4')
define(`confDEF_USER_ID', `8:8')
define(`confDOMAIN_NAME', `smtp.test.local')

OSTYPE(linux)
DOMAIN(generic)

undefine(`UUCP_RELAY')
undefine(`BITNET_RELAY')

define(`MILTER', `1')
INPUT_MAIL_FILTER(`php-milter1', `S=local:/var/milter/php-
milter1.sock, F=R')
INPUT_MAIL_FILTER(`php-milter2', `S=local:/var/milter/php-
milter2.sock, F=T, T=S:10m;R:10m;E:10m')
INPUT_MAIL_FILTER(`php-milter3', `S=inet:999@localhost')

define(`confINPUT_MAIL_FILTER', `php-milter2, php-milter1, php-
milter3')
```

Listing 5: `sendmail.mc`

The lines of interest start after the `define(`MILTER', `1')` command, which tells sendmail to enable Milter. The following lines define the actual filters, the first two communicate via Unix-domain sockets, the third uses an IP socket on port 999.

The last line is not necessarily required. It specifies the order in which the filters should be applied, if left out the filters are applied in the order they are declared.

Back to the lines where the filters are defined, the possible flags `F=` for `INPUT_MAIL_FILTER()` define the fallback action in case a filter is not available. Possible actions are listed in figure 1.

Flag	Meaning
------	---------

Setup and Configuration

R	Reject connection if filter unavailable
T	Temporary fail connection if filter unavailable

Figure 1: Fallback actions

If not specified otherwise Sendmail uses default timeouts when waiting for a filters response. These can be overridden using the T= parameter with the following flags:

Flag	Meaning
S	Timeout for sending information from the MTA to a filter. (default: 10s)
R	Timeout for reading reply form the filter. (default: 10s)
E	Overall timeout between sending end-of-message to filter and waiting for the final acknowledgement. (default: 5m)

Figure 2: Filter timeouts

The configuration is done and all that is left is the need to start the filters. The following line should do to the start for the filter milter.php, note that for every filter defined in the configuration file, there should be a separate socket:

```
php-milter -D -p /var/milter/php-milter1.sock milter.php
```

Listing 6: start milter

The path given in `-p` should be the same as the path defined in the Sendmail configuration file.

To get all these and further info about the php-milter just run it with the `-h` flag. This will provide you with the following output:

```
Usage: php-milter [options] [-f] <file> [args...]  
       php-milter [options] -r <code> [args...]  
       php-milter [options] [-- args...]  
-a                Run interactively  
-c <path>|<file> Look for php.ini file in this directory  
-n                No php.ini file will be used  
-d foo[=bar]     Define INI entry foo with value 'bar'  
-D                run as daemon  
-e                Generate extended information for  
debugger/profiler  
-f <file>        Parse <file>.  
-h                This help  
-p <socket>      path to create socket  
-v                Version number  
-V <n>           set debug level to n (1 or 2).  
-z <file>        Load Zend extension <file>.
```

```
args... Arguments passed to script. Use -- args when
first argument starts with - or script is read from stdin
```

Listing 7: all flags for php-milter

1.3 milter sapi

Before presenting an actual example, let's have a look at a list of all possible callbacks, what they do and their return values.

1.3.1 Callbacks

All callbacks are listed in Figure 3, for now only the short description should be enough when exactly they are called is shown later on in the example.

Function	Description
milter_init	Initialize filter.
milter_connect	connection info
milter_helo	SMTP HELO/EHLO command
milter_envrcpt	envelope recipient
milter_header	header
milter_eoh	end of header
milter_body	body
milter_eom	end of message
milter_abort	message aborted
milter_close	connection cleanup

Figure 3: Callbacks in milter

1.3.2 Init Flags

In order to allow a filter to take the desired actions, i.e. adding headers, it needs to be initialized in milter_init with the correct flags.

Flag	Description
SMFIF_ADDHDRS	This filter may add headers.
SMFIF_CHGHDRS	This filter may change and/or delete headers.

SMFIF_CHGBODY	This filter may replace the body during filtering. (significant performance impact)
SMFIF_ADDRcpt	This filter may add recipients to the message.
SMFIF_DELRcpt	This filter may remove recipients from the message.

Figure 4: initialization flags

1.3.3 Message Modification Functions

Message modification functions are only valid if they are called from within `milter_eom()`. Each function returns either `TRUE` or `FALSE` to indicate the status of the operation.

Function	Description
<code>smfi_addheader(string headerf, string headerv)</code>	Add a header to the message.
<code>smfi_chgheader(string headerf, string headerv)</code>	Change or delete a header.
<code>smfi_addrcpt(string rcpt)</code>	Add a recipient to the envelope.
<code>smfi_delrcpt(string rcpt)</code>	Delete a recipient from the envelope.
<code>smfi_replacebody(string body)</code>	Replace the body of the message.

Figure 5: message modification functions

1.3.4 Data Access Functions

Besides the above mentioned message modification functions that can only be called in `milter_eom()` there are data access functions that can be called from within any callback.

Function	Description
<code>smfi_getsymval(string macro)</code>	Returns the value of the given macro or <code>NULL</code> if the macro is not defined.
<code>smfi_setreply(string rcode, string xcode, string message)</code>	Directly set the SMTP error reply code for this connection.

Figure 6: data access functions

1.3.5 Return Values

All the callbacks return SMFIS_CONTINUE by default, but can return any of the return values in Figure 7. Each routine in milter belongs to one of the following groups:

- recipient-oriented – affects a single message recipient
- message-oriented – affects a single message
- connection-oriented – affects an entire connection

depending on what group a routine belongs to milter reacts on the given return value.

milter_envrcpt is recipient-oriented, milter_connect, milter_helo and milter_close are connection-oriented, the rest is message-oriented.

Return value	Description
SMFIS_CONTINUE	Default. Continue processing.
SMFIS_REJECT	For a connection-oriented routine, reject this connection; call milter_close. For a message-oriented routine (except milter_eom or milter_abort), reject this message. For a recipient-oriented routine, reject the current recipient (but continue the current message).
SMFIS_DISCARD	For a message- or recipient-oriented routine, accept this message, but silently discard it. Should not be returned by a connection-oriented routine.
SMFIS_ACCEPT	For a connection-oriented routine, accept this connection without further filter processing; call milter_close. For a message- or recipient-oriented routine, accept this message without further filtering.
SMFIS_TEMPFAIL	Return a temporary failure. For message-oriented routine, except milter_envfrom, fail for this message. For a connection-oriented routine, fail for this connection, call milter_close. For a recipient-oriented routine, only fail for the current recipient; continue message processing

Figure 7: return values

1.4 Example

Now it's time to have a look at the source of a filter written in PHP4. The following filter is quite exhaustive and covers all callbacks that are available through the milter sapi to demonstrate the full potential. The only use of this filter is to write each action into a logfile to get feedback on if and when a callback was executed. If you would like to try out this example for yourself, it is accessible on the internet at <http://cvs.php.net/co.php/php4/sapi/milter/milter.php>

```
function milter_log($msg)
{
    $GLOBALS['log'] = fopen("/tmp/milter.log", "a");
    fwrite($GLOBALS['log'], date("[H:i:s d.m.Y]") . "\t{$msg}\n");
    fclose($GLOBALS['log']);
}
```

All that our example should do is to write an entry into a logfile whenever a function is called. That's what this first function does.

```
function milter_init() {
    milter_log("-- startup --");
    milter_log("milter_init()");
    smfi_setflags(SMFIF_ADDHDRS);
}
```

This function is called once on sapi startup, here you can specify the action the filter may take. There are different possibilities, as for example SMFIF_ADDHDRS, which allows the filter to add headers.

Be aware that the php-filter file could be changed without restarting php-milter, but milter_init() would not be called again, so if the newer version of the php-filter file sets new smfi flags php-milter needs to be restarted.

```
function milter_connect($connect)
{
    milter_log("milter_connect('$connect')");
}
```

This function is called for every new SMTP connection.

```
function milter_helo($helo)
{
    milter_log("milter_helo('$helo')");
}
```

Called whenever the client sends a HELO/EHLO command.

Filtering Mail with php-milter

```
function milter_envfrom($args)
{
    milter_log("milter_envfrom(args[])");
    foreach ($args as $ix => $arg) {
        milter_log("\targs[$ix] = $arg");
    }
}
```

Is called once at the beginning of each message, before `milter_envrcpt`. `args[0]` is guaranteed to be the sender address. Later arguments are the ESMTP arguments.

```
function milter_envrcpt($args)
{
    milter_log("milter_envrcpt(args[])");
    foreach ($args as $ix => $arg) {
        milter_log("\targs[$ix] = $arg");
    }
}
```

Is called once per recipient, hence one or more times per message and immediately after `milter_envfrom`. `$args` contains null-terminated SMTP command arguments, `$args[0]` is guaranteed to be the recipient address. Later arguments are the ESMTP arguments.

```
function milter_header($header, $value)
{
    milter_log("milter_header('$header', '$value')");
}
```

Is called zero or more times between `milter_envrcpt` and `milter_eoh` and once per message header.

```
function milter_eoh()
{
    milter_log("milter_eoh()");
}
```

Is called once after all headers have been sent and processed.

```
function milter_body($bodypart)
{
    milter_log("milter_body('$bodypart')");
}
```

Is called zero or more times between `milter_eoh` and `milter_eom`. Since bodies can be very large this callback can significantly impact filter performance.

Outlook

```
function milter_eom()
{
    milter_log("milter_eom()");
    /* add PHP header to the message */
    smfi_addheader("X-PHP", phpversion());
}
```

Is called once after calls to `milter_body` for a given message. Most of the api functions, that alter the message can only be called within this callback.

```
function milter_abort()
{
    milter_log("milter_abort()");
}
```

May be called at any time during message processing, i.e. between some message-oriented routine and `milter_eom`.

```
function milter_close()
{
    milter_log("milter_close()");
}
```

Is always called once at the end of each connection.

1.5 Outlook

At present time there has not been made any testing regarding performance, it can therefore not be said what load of emails could be handled with `php-milter`.

It should also be mentioned that this implementation is still experimental, it is feature complete, but still contains some memory leaks.