Incomplete mapSoN User Manual

Peter Simons

simons@computer.org

1. Introduction

The amount of unsolicited commercial e-mail ("spam") circulating in the Internet today has become unbearable for most people. Many approaches have been proposed to stop this junk from filling up your mailboxes, such as the Real-time blackhole list (http://mail-abuse.org/rbl+/), Teergruben (http://www.iks-jena.de/mitarb/lutz/usenet/teergrube.en.html), or procmail-based anti-spam recipts (http://junkfilter.zer0.org/).

mapSoN is another anti-spam system, but it uses an approach entirely different than those systems named before. Instead of trying to recognize spam by the IP address of the SMTP dialag's peer or by certain patters in the mail's body, mapSoN uses the sender's e-mail address to decide whether the e-mail is delivered to your mailbox or not: Any e-mail that comes from a "known" address may pass, any e-mail that comes from an e-mail address seen for the first time needs special confirmation, before it may pass.

"Special confirmation" means that mapSoN will generate an MD5 checksum of the to-be-confirmed mail and stores the mail in a temporary spool directory. Then it sends an *request for confirmation* to the address from which the mail was coming from. In this request, it will include the MD5 checksum and ask the recipient to reply back and to quote that MD5 hash. Once mapSoN sees that MD5 hash again, it considers that a confirmation of the original mail, delivers the deferred mail from the spool to your mailbox, and adds the sender's address to the database of known addresses, so that the next time he tries to contact you, his mail will pass through mapSoN immediately..

This heuristic catches almost any spam mail, because spammers have to fake their sender addresses in order to avoid being held responsible for their abuse. Hence, their address will most likely not be in the database of known addresses, nor will they ever receive the request for confirmation mail!

This manual describes how to compile, install, and configure mapSoN in various setups.

2. Building mapSoN

Compiling the software should be pretty straight forward; all you'll have to do is the usual routine:

```
./configure
make
```

```
make install
```

The last step may require super-user privileges, so have the root password ready if you're going for a system-wide installation.

Be adviced that mapSoN needs a fairly recent C++ compiler because it make full use of the new ISO C++ language features. If you're using the GNU C Compiler (http://gcc.gnu.org/) version 2.95 or later, you won't have any problems. But other compilers are known not to be ISO C++ compatible. If you're having trouble, send me an e-mail and I'll see what I can do.

The `configure` script is a standard GNU Autoconf (http://www.gnu.org/software/autoconf/) script, which supports all the usual options. If you are not familiar with these scripts, please refer to the "Running `configure` Scripts" section of the Autoconf user manual, which is available at:

http://www.gnu.org/manual/autoconf/html_chapter/autoconf_13.html#SEC129

The following list will show only those options that are specific to mapSoN.

`-with-mailboxdir=`*`DIR`*

In order to deliver incoming e-mail to your mailbox, mapSoN needs to know where the user mailboxes are located on your system. It tries to figure that out automatically by checking for the existance of the directories `/var/mail` and `/var/spool/mail`, but depending on your setup, you might want to choose another path here.

`-with-mta=`*`PATH`*

mapSoN needs to know the complete path to your system's mail transport agent (MTA) in order to send out requests for confirmation. The `configure` script will assume that you have sendmail installed and look for it in various locations, but you can (and may have to) set the right choice manually using this option.

`-with-debug`

Per default, mapSoN comes with a couple of additional debug messages, which you can enable on the command line or in the configuration file if you feel something's going wrong. But in order for these log messages to be available, the binary must have been compiled with the DEBUG flag. Using this option, you can manually decide whether you want these log messages compiled in or not.

Any of the paths you configure here are only used as defaults. You can override them at run-time in the configuration file mapSoN reads at startup.

3. Command Line Syntax

mapSoN understans severel optional parameters an the command line, which allow you to override the compiled-in default or the settings in the config file. The standard Unix synopsis line is:

mapson [-d | –debug] [-c `config` | –config-file `config`] [`mail`...]

Here is a list of all options together with a short description of what the respective option does:

`-h`

Show mapSoN usage information.

`-d`
`-debug`

Enable debugging. Please note that debugging is only available if mapSoN has been compiled with the define DEBUG. Otherwise, the debug code is not included in the binary.

`-c` *`config`*
`-config-file` *`config`*

Use the configuration file *`config`* rather the default.

`mail ...`

If any parameter is specified on the command line that is not an option, mapSoN will go into *gather addresses* mode. The parameters are interpreted as filenames, each of the files containing an e-mail that mapSoN will parse. Any sender address mapSoN finds in these mails will be added to the database of known addresses. This mode is meant to import addresses from your mail archive to the database.

4. The mapSoN Configuration File

The configuration file may contain the following directives:

`Mailbox` *`file`*

This directive sets the complete path of the mailbox file, where mapSoN stores approved mails.

`SpoolDir` *`directory`*

This directive sets the complete path to the directory, in which deferred mails will be spooled until a confirmation arrives for them.

`AddressDB` *`file`*

This directive sets the complete path of the file mapSoN uses to store the "known" addresses.

`ReqConfirmTemplate` *`file`*

This directive sets the complete path to the request-for-confirmation template file mapSoN uses to generate the request-for-confirmation mail sent to first-time originators.

An arbitrary number of alternate paths can be specified, if they're separated by colons, for example:

```
$HOME/.mapson/reqmail.template:$DATADIR/reqmail.template:...
```

In this setup, mapSoN would first try to load the file `$HOME/.mapson/reqmail.template`. If that failed, it would try `$DATADIR/reqmail.template`, and so on, until one of the files can be loaded successfully.

This is an extreme useful feature if you are a system administrator who wishes to allow all users of the system to use mapSoN without having to create a request-for-confirmation template of their own: Configure mapSoN to load that request-for-confirmation template first, that is located in the user's home directory. If this file does not exist, then fall back to the system-wide file.

In effect, that means that the user can simply use mapSoN to filter his mail, and if he ever feels like it, he can create a request-for-confirmantion template file of his own and it will be preferred over the system-wide one.

`MTA` *`command`*

This directive sets the command mapSoN will use to send-out a request-for-confirmation mail. The actual mail will be piped into the started process.

`PassIncorrectMails` *`boolean`*

When mapSoN parses the incoming mail's headers for the addresses, it may detect syntax errors in the mail header, that do not cause a fatal error, but that surely hint to the fact that this mail was not created by an RFC822-conformant mail client.

Many spam mails contain incorrect header lines, so you may chose to have mapSoN fail on *any* syntax error – even non-fatal ones. "Failing" means that mapSoN will abort and return the return code configured below to the MTA. Depending on the setting of the return code, the MTA will then bounce the mail.

The parameter given to this option is a boolean, meaning that you may specify either `yes` or `no`.

`StrictRFCParser` *`boolean`*

If you enable this option by specifying `yes`, mapSoN will perform additional syntax checks on the incoming mail, if you say `no`, it will check only those headers that are needed for mapSoN to operate at all.

Enabling this option makes little sense unless you disable the PassIncorrectMails option.

`RuntimeErrorRC` *`integer`*

This directive sets the return code mapSoN exits with in case it had to abort with a run-time error. Possible run-time errors are failure to open file, lack of available memory, etc. . . .

The default choice is "75", which sendmail will interpret as a temporary system error, so it will queue the mail and re-try.

A valid return code is a positive integer up to 128.

`SyntaxErrorRC` *`integer`*

This directive sets the return code mapSoN exits with in case it encountered a fatal syntax error in the e-mail. If PassIncorrectMails is disabled, non-fatal syntax errors will also cause mapSoN to abort with this return code.

The default choice is "65", which sendmail will interpret as a permanent error that causes the mail to bounce.

A valid return code is a positive integer up to 128.

`Debug` *`boolean`*

If you enable debugging messages by saying `yes` here, mapSoN will log additional information about its procssing of the mail. If you say `no`, mapSoN will log only very few messages at all.

Debugging is available only when the binary has been compiled with the DEBUG symbol defined. Currently, that is the default, though, so unless you exclicitely disabled it, debugging will be available.

In order to make the contents of the configuration file as independent from the system's directory structure as possible, mapSoN provides a set of environment variables, which are guaranteed to be defined. You can use them anywhere in the data part of a configuration directive, and you can use the usual manipulations on them.

Envirenment variables are looked-up case-sensitive, so `$home` is not the same thing as `$HOME`. This behavior is different in the request-for-confirmation template, where you can spell the variables upper- or lower-case as you wish. That's because the variables there are not coming from the environmnet, but are mapSoN's internal variables. So be sure not to confuse that, because an undefined variable in this file will cause mapSoN to abort with an error.

mapSoN will not overwrite already existing variables, though! If your system defines, for example, the `$HOME` variable, then you'll get the value from the system's variable.

Here is the complete list:

`$MAILBOXDIR`

This variable contains the complete path of directory, in which the system's mailboxes are located, usually `/var/spool/mail`. Please note that the value provided here is the one determined at *compile-time*, so if you changed your system's installation and want to rely on this variable, you'll have to re-compile.

`$MTA`

This variable contains the path to the systems mail transport agent. Please note, that this is only the path of the executable – for example `/usr/sbin/sendmail` –, the variable does not contain the flags that must be passed to the MTA in order to do something useful.

`$DATADIR`

This variable contains the complete path of the directory, which has been compiled into mapSoN as the directory where read-only architecture-independent data should be stored. You will, for example, find the system-wide request-for-confirmation template file here.

`$USER`

This variable contains the name of the user under which mapSoN is running. Depending on your MTA, this must not necessarily be the user who is receiving mail! If you're using sendmail, though, you're on the secure side.

`$HOME`

This variable contains the complete path of `$USER`'s home directory.

5. The Request-for-Confirmation File

The request-for-confirmation mail is created by loading the configured template file and exanding the variables contained in it. The result is then piped into the command, you have configured as `MTA`.

The following variables are provided in the request-for-confirmation mail template file:

`$md5hash`

`$envelope`

`$sender`

`$return_path`

`$header`

`$body`

`$messageid`

In addition to those, the following arrays are provided:

`$headerlines[]`

`$header[]`

`$body[]`

6. How to Activate mapSoN

Before you can activate mapSoN you have to make sure that all required directories exist and that all required files can be created. mapSoN will try to create the address database file, for instance, but the default setting is that this database is located at `$HOME/.mapon/address-db`, and mapSoN *will not* create the `$HOME/.mapson` directory.

The same applies to the mail spool directory, where mapSoN stores those mails that need a confirmation before they are delivered. The default location is `$HOME/.mapson/spool`, and mapSoN will create the spool directory itself, but it won't create the `$HOME/.mapson` directory.

The cut a long story short: Create the directory `.mapson` in your home directory before you try to use mapSoN. If you changed the configuration, though, the locations may vary.

Once that's out of the way, you need to create a template file that mapSoN will use to generate the request-for-confirmation mail. The installation process has created you a sample file for this purpose to `/usr/local/share/mapson/reqmail.template-sample`, so thats probably a good starting point.

Then, all that's left to do is to hook mapson into your mail system so that incoming e-mails are no longer delivered to your mailbox directly but are piped into mapSoN. How you can do this depends on the setup you have. The following sections will give you detailed instructions for the different scenarios.

6.1. Via `.forward`

If you're using a plain sendmail installation, and don't have any fancy local mailers like procmail installed, you can activate mapSoN by writing an appropriate pipe-command to the file `.forward` in your home directory:

```
"|exec /usr/local/bin/mapson"
```

Apparently, on some systems the home directory must grant execute-permission to "other" for the MTA to evaluate that file. So if you created the `.forward` file as shown above and still there's no sign of any mapSoN activity, execute **`chmod 711 $HOME`** and try again.

Another potential obstacle is that some sendmail installation use the restricted shell (smrsh) for the execution of the local mailer. This shell will not allow users to execute arbitrary commands in the `.forward` file. If your system uses the smrsh, you must create a link from `/usr/local/bin/mapson` to `/usr/adm/sm.bin/mapson` in order to enable mapSoN. (The paths may vary from system to system, obviously.)

6.2. Via `.procmailrc`

Most systems these days use procmail to deliver local mail. That means, that you can configure your local mailer by adding cryptic recipts in an entirely undocumented syntax to the file `.procmailrc` in your home directory.

The good thing about procmail is that it gives you way more control over which e-mail is going where. For example, you don't want to pipe mail through mapSoN that has been delivered to you via a mailing list: It would be really gross to request confirmation for a mail from someone who doesn't even know you and hasn't even mailed you – all he did was to post on a mailing list.

Unfortunately, recognizing mail that came from a mailing list is not an easy thing to do, but the following procmail recipt should work pretty good:

```
# Pipe anything, that is not a reply or comes from a mailing
# list, into mapson for approval.
#
:0 w
* !^(In-Reply-To|References|Message-Id):.*example.com
* !^Precedence: (list|bulk|junk)
* !^Auto-Submitted:
| /usr/local/mapson/bin/mapson
```

Using procmail's "argument" feature, you can make this work even better, but that is still marked TODO in this manual . . . Here are the recipts, nonetheless, in case you think you know what you're doing.

```
ARGUMENT="$1"

# An argument of 'failsafe' will always by-pass all other
# rules to make sure that I have a reliably working address
# no matter what happens here.
#
:0
* ARGUMENT ?? failsafe
$DEFAULT

# Confirmation mails go into mapSoN.
# Note: Not all systems support that regular expression syntax!
#
:0
* ARGUMENT ?? [a-f0-9]{32}
| /usr/local/mapson/bin/mapson
:0 ec

# Anything that has such an argument is probably coming from
# a mailing list and should not go through mapSoN.
#
:0
* ARGUMENT ?? ..*
$DEFAULT
```

7. Expiring the Mail Spool

8. Importing Addresses From a Mail Archive

Of course it would be unpolite to have mapSoN send out requests for confirmation to people who you have been communicating with you for months or years, just because you installed a new tool. If you were wise enough to archive your old e-mails, there's a simple way to avoid that happening: Import their addresses into mapSoN's database.

Unfortunately, most mail readers archive old mails in one single file: Each new mail is just appended at the end, just like the mailbox format itself. Currently, mapSoN can not deal with those files. The current version can import addresses only from an archive where a each mail is stored in a separate file, like the archives maintained by the Gnus software, that is part of Emacs, for example.

In this case, though, it's simple enough: Just start mapSoN and give it the file names as parameters on the command line. You might want to enable debugging by giving it the `-d` flag, so that you can see what's going on:

```
simons@peti:~/mail-archive$ mapson -d *
1:
12:
    simons@peti.gmd.de................................ new
16:
    simons@peti.gmd.de................................ known
17:
    th@example.com.................................... new
    th@example.com.................................... known
    th@example.com.................................... known
19:
    bscw@cscwmail.example.org......................... new
    manfred.bogen@gmd.example.org..................... new
53:
    pakhomenko@example.com............................ new
    pakhomenko@example.com............................ known
```

Depending on the size of your mail archive, this may take a while, but usually mapSoN is pretty quick.

Once that's finished, you'll have a pretty good database to start with, and it's highly unlikely that someone, who has been in contact with you before, will be bothered with an request-for-confirmation mail.

A. License

This software is copyrighted by Peter Simons <`simons@computer.org`>. Permission is granted to use it under the terms of the GNU General Public License. For further details, refer to the file `LICENSE` included in the software distribution or see http://www.gnu.org/licenses/gpl.html in case that file is missing.

mapSoN uses the "Variable Expression Library", which is included in the distribution for comfort. This library is *not* part of the mapSoN package and is licensed under the terms described in the file `libvarexp/LICENSE`. Should that file be missing in your distribution, contact me for a copy.

The MD5 library included in this distribution has been taken from the "GNU C Library" and is licenced under the terms of the GNU Library General Public License. The licence file can be found in the file `libmd5/COPYING.LIB`.

The `getopt()` and `getopt_long()` implementation used my mapSoN in case the routine are not available in the system libraries come from the GNU C Library, too, and are licensed under the terms of the GNU Lesser General Public License, what is effectively just a newer version of the library-licence version. A copy of the license can be found in `libgetopt/COPYING`.

B. The Default Configuration

```
Mailbox ${MAILBOXDIR}/${USER}
SpoolDir ${HOME}/.mapson/spool
AddressDB ${HOME}/.mapson/address-db
ReqConfirmTemplate ${HOME}/.mapson/reqmail.template:${DATADIR}/reqmail.template
MTA ${MTA} '-f&lt;&gt;' -i -t
PassIncorrectMails true
StrictRFCParser false
RuntimeErrorRC 75
SyntaxErrorRC 65
Debug false
```

C. A Request-for-Confirmation Template Example

```
From: username+${MD5HASH}@example.com (Real Name's Anti-Spam-Tool)
To: ${ENVELOPE:-${RETURN_PATH:-${SENDER}}}
Subject: please confirm / bitte bestaetigen
Precedence: junk
Auto-Submitted: auto-generated
References: $MESSAGEID
In-Reply-To: $MESSAGEID
```

```
               English                                 Deutsch

Because I receive several dozen spam    Da ich am Tag dutzende Spam-Mails
mails each day, I installed a small     erhalte, habe ich ein Programm
tool that will defer incoming e-mail    installiert, das jede eingehende
if it comes from an address unknown     E-Mail abweist, wenn diese von einer
to me. This is the case with the mail   mir unbekannten Absendeadresse
you sent me, I'm afraid.                kommt. Leider ist dies bei Ihrer
                                        E-Mail der Fall.
Before your mail will be delivered to
my mailbox, I need a confirmation,      Bevor ich die E-Mail tatsaechlich
that your sender address is a valid     zugestellt bekomme, brauche ich eine
e-mail address. You can confirm this    Bestaetigung, dass die Absendeadresse
simply by replying to this e-mail --    "echt" ist. Sie koennen dies einfach
the contents of the reply doesn't       dadurch bestaetigen, dass Sie auf
matter. Once my tool sees your reply,   diese Mail antworten. Der Inhalt der
it will deliver the original mail to    Antwort spielt dabei keine Rolle, es
my mailbox and I'll answer it as soon   geht nur darum, dass das Programm
as possible.                            sieht, dass Ihre Adresse nicht
                                        gefaelscht war.
I am sorry that this is somewhat
annoying for you, but the amount of     Ich bedauere, dass ich Ihnen damit
unsolicited commercial advertisement    Umstaende bereiten muss, aber die
sent through the Internet these days    Menge an unerwuenschter Werbung, die
is to much for me to bear.              derzeit durch das Internet geistert,
                                        war einfach zuviel.

Your e-mail was:                        Ihre E-Mail war:

[   | ${HEADER[#]}]   |
[${BODY[#]:+   | }${BODY[#]}]{0,5}   | \[...\]
```