

Section 40 - Content scanning at ACL time

40. Content scanning at ACL time

The extension of Exim to include content scanning at ACL time, formerly known as “exiscan”, was originally implemented as a patch by Tom Kistner. The code was integrated into the main source for Exim release 4.50, and Tom continues to maintain it. Most of the wording of this chapter is taken from Tom’s specification.

It is also possible to scan the content of messages at other times. The `local_scan()` function (see chapter 41) allows for content scanning after all the ACLs have run. A transport filter can be used to scan messages at delivery time (see the `transport_filter` option, described in chapter 24).

If you want to include the ACL-time content-scanning features when you compile Exim, you need to arrange for `WITH_CONTENT_SCAN` to be defined in your `Local/Makefile`. When you do that, the Exim binary is built with:

-

Two additional ACLs (`acl_smtp_mime` and `acl_not_smtp_mime`) that are run for all MIME parts for SMTP and non-SMTP messages, respectively.

-

Additional ACL conditions and modifiers: `decode`, `malware`, `mime_regex`, `regex`, and `spam`. These can be used in the ACL that is run at the end of message reception (the `acl_smtp_data` ACL).

-

An additional control feature (“`no_mbox_unspool`”) that saves spooled copies of messages, or parts of messages, for debugging purposes.

-

Additional expansion variables that are set in the new ACL and by the new conditions.

-

Two new main configuration options: `av_scanner` and `spamd_address`.

There is another content-scanning configuration option for `Local/Makefile`, called `WITH_OLD_DEMIME`. If this is set, the old, deprecated `demime` ACL condition is compiled, in addition to all the other content-scanning features.

Content-scanning is continually evolving, and new features are still being added. While such features are still unstable and liable to incompatible changes, they are made available in Exim by setting options whose names begin `EXPERIMENTAL_` in `Local/Makefile`. Such features are not documented in this manual. You can find out about them by reading the file called `doc/experimental.txt`.

All the content-scanning facilities work on a MBOX copy of the message that is temporarily created in a file called: `<spool_directory>/scan/<message_id>/<message_id>.eml`

The `.eml` extension is a friendly hint to virus scanners that they can expect an MBOX-like structure inside that file. The file is created when the first content scanning facility is called. Subsequent calls to content scanning conditions open the same file again. The directory is recursively removed when the `acl_smtp_data` ACL has finished running, unless `control = no_mbox_unspool`

has been encountered. When the MIME ACL decodes files, they are put into the same directory by default.

40.1 Scanning for viruses

The `malware` ACL condition lets you connect virus scanner software to Exim. It supports a “generic” interface to scanners called via the shell, and specialized interfaces for “daemon” type virus scanners, which are resident in memory and thus are much faster.

You can set the `av_scanner` option in first part of the Exim configuration file to specify which scanner to use, together with any additional options that are needed. The basic syntax is as follows: `av_scanner = <scanner-type>:<option1>:<option2>:[...]`

If you do not set `av_scanner`, it defaults to `av_scanner = sophie:/var/run/sophie`

If the value of `av_scanner` starts with dollar character, it is expanded before use. The following scanner types are supported in this release: `aveserver`

This is the scanner daemon of Kaspersky Version 5. You can get a trial version at <http://www.kaspersky.com>. This scanner type takes one option, which is the path to the daemon's UNIX socket. The default is shown in this example: `av_scanner = aveserver:/var/run/aveserver`
`clamd`

This daemon-type scanner is GPL and free. You can get it at <http://www.clamav.net/>. Some older versions of `clamd` do not seem to unpack MIME containers, so it used to be recommended to unpack MIME attachments in the MIME ACL. This no longer believed to be necessary. One option is required: either the path and name of a UNIX socket file, or a hostname or IP number, and a port, separated by space, as in the second of these examples: `av_scanner = clamd:/opt/clamd/socket`
`av_scanner = clamd:192.168.2.100 1234`

If the option is unset, the default is `/tmp/clamd`. Thanks to David Saez for contributing the code for this scanner. `cmdline`

This is the keyword for the generic command line scanner interface. It can be used to attach virus scanners that are invoked from the shell. This scanner type takes 3 mandatory options:

-

The full path and name of the scanner binary, with all command line options, and a placeholder (`%s`) for the directory to scan.

-

A regular expression to match against the `STDOUT` and `STDERR` output of the virus scanner. If the expression matches, a virus was found. You must make absolutely sure that this expression matches on `“virus found”`. This is called the `“trigger”` expression.

-

Another regular expression, containing exactly one pair of parentheses, to match the name of the virus found in the scanners output. This is called the `“name”` expression.

For example, Sophos Sweep reports a virus on a line like this: `Virus 'W32/Magistr-B' found in file ./those.bat`

For the trigger expression, we can just match the word `“found”`. For the name expression, we want to extract the `W32/Magistr-B` string, so we can match for the single quotes left and right of it. Altogether, this makes the configuration setting: `av_scanner = cmdline:\`

```

/path/to/sweep -all -rec -archive %s:\
found:'(.+)'
```

`drweb`

The DrWeb daemon scanner (<http://www.sald.com/>) interface takes one argument, either a full path to a UNIX socket, or an IP address and port separated by white space, as in these examples: `av_scanner = drweb:/var/run/drwebd.sock`
`av_scanner = drweb:192.168.2.20 31337`

If you omit the argument, the default path `/usr/local/drweb/run/drwebd.sock` is used. Thanks to Alex Miller for contributing the code for this scanner. `fsecure`

The F-Secure daemon scanner (<http://www.f-secure.com>) takes one argument which is the path to a UNIX socket. For example: `av_scanner = fsecure:/path/to/.fsav`

If no argument is given, the default is `/var/run/.fsav`. Thanks to Johan Thelmen for contributing the code for this scanner. `kavdaemon`

This is the scanner daemon of Kaspersky Version 4. This version of the Kaspersky scanner is outdated. Please upgrade (see `aveserver` above). This scanner type takes one option, which is the path to the daemon's UNIX socket. For example: `av_scanner = kavdaemon:/opt/AVP/AvpCtl`

The default path is `/var/run/AvpCtl. mksd`

This is a daemon type scanner that is aimed mainly at Polish users, though some parts of documentation are now available in English. You can get it at <http://linux.mks.com.pl/>. The only option for this scanner type is the maximum number of processes used simultaneously to scan the attachments, provided that the demime facility is employed and also provided that mksd has been run with at least the same number of child processes. For example: `av_scanner = mksd:2`

You can safely omit this option (the default value is 1). `sophie`

Sophie is a daemon that uses Sophos's libsavi library to scan for viruses. You can get Sophie at <http://www.vanja.com/tools/sophie/>. The only option for this scanner type is the path to the UNIX socket that Sophie uses for client communication. For example: `av_scanner = sophie:/tmp/sophie`

The default path is `/var/run/sophie`, so if you are using this, you can omit the option.

When `av_scanner` is correctly set, you can use the malware condition in the DATA ACL. Note: You cannot use the malware condition in the MIME ACL.

The `av_scanner` option is expanded each time malware is called. This makes it possible to use different scanners. See further below for an example. The malware condition caches its results, so when you use it multiple times for the same message, the actual scanning process is only carried out once. However, using expandable items in `av_scanner` disables this caching, in which case each use of the malware condition causes a new scan of the message.

The malware condition takes a right-hand argument that is expanded before use. It can then be one of

-

`"true"`, `"*"`, or `"1"`, in which case the message is scanned for viruses. The condition succeeds if a virus was found, and fail otherwise. This is the recommended usage.

-

`"false"` or `"0"`, in which case no scanning is done and the condition fails immediately.

-

A regular expression, in which case the message is scanned for viruses. The condition succeeds if a virus is found and its name matches the regular expression. This allows you to take special actions on certain types of virus.

You can append `/defer_ok` to the malware condition to accept messages even if there is a problem with the virus scanner.

When a virus is found, the condition sets up an expansion variable called `$malware_name` that contains the name of the virus. You can use it in a message modifier that specifies the error returned to the sender, and/or in logging data.

If your virus scanner cannot unpack MIME and TNEF containers itself, you should use the demime condition (see section 40.6) before the malware condition.

Here is a very simple scanning example: `deny message = This message contains malware ($malware_name)`

```
demime = *
malware = *
```

The next example accepts messages when there is a problem with the scanner: `deny message = This message contains malware ($malware_name)`

```
demime = *
malware = */defer_ok
```

The next example shows how to use an ACL variable to scan with both sophie and avserver. It assumes you have set: `av_scanner = $acl_m0`

in the main Exim configuration. `deny message = This message contains malware ($malware_name)`

```
set acl_m0 = sophie
malware = *
```

```
deny message = This message contains malware ($malware_name)
set acl_m0 = aveserver
malware = *
40.2 Scanning with SpamAssassin
```

The spam ACL condition calls SpamAssassin's spamd daemon to get a spam score and a report for the message. You can get SpamAssassin at <http://www.spamassassin.org>, or, if you have a working Perl installation, you can use CPAN by running: `perl -MCPAN -e 'install Mail::SpamAssassin'`

SpamAssassin has its own set of configuration files. Please review its documentation to see how you can tweak it. The default installation should work nicely, however.

After having installed and configured SpamAssassin, start the spamd daemon. By default, it listens on 127.0.0.1, TCP port 783. If you use another host or port for spamd, you must set the `spamd_address` option in the global part of the Exim configuration as follows (example): `spamd_address = 192.168.99.45 387`

You do not need to set this option if you use the default. As of version 2.60, spamd also supports communication over UNIX sockets. If you want to use these, supply `spamd_address` with an absolute file name instead of a address/port pair: `spamd_address = /var/run/spamd_socket`

You can have multiple spamd servers to improve scalability. These can reside on other hardware reachable over the network. To specify multiple spamd servers, put multiple address/port pairs in the `spamd_address` option, separated with colons: `spamd_address = 192.168.2.10 783 : \
192.168.2.11 783 : \
192.168.2.12 783`

Up to 32 spamd servers are supported. The servers are queried in a random fashion. When a server fails to respond to the connection attempt, all other servers are tried until one succeeds. If no server responds, the spam condition defers.

Warning: It is not possible to use the UNIX socket connection method with multiple spamd servers. 40.3 Calling SpamAssassin from an Exim ACL

Here is a simple example of the use of the spam condition in a DATA ACL: `deny message = This message was classified as SPAM
spam = joe`

The right-hand side of the spam condition specifies the username that SpamAssassin should scan for. If you do not want to scan for a particular user, but rather use the SpamAssassin system-wide default profile, you can scan for an unknown user, or simply use `“nobody”`. However, you must put something on the right-hand side.

The username allows you to use per-domain or per-user antispam profiles. The right-hand side is expanded before being used, so you can put lookups or conditions there. When the right-hand side evaluates to `“0”` or `&ldquo>false”`, no scanning is done and the condition fails immediately.

Scanning with SpamAssassin uses a lot of resources. If you scan every message, large ones may cause significant performance degradation. As most spam messages are quite small, it is recommended that you do not scan the big ones. For example: `deny message = This message was classified as SPAM
condition = ${if < {$message_size}{10K}}
spam = nobody`

The spam condition returns true if the threshold specified in the user's SpamAssassin profile has been matched or exceeded. If you want to use the spam condition for its side effects (see the variables below), you can make it always return `&ldquo>true”` by appending `:true` to the username.

When the spam condition is run, it sets up a number of expansion variables. With the exception of `$spam_score_int`, these are usable only within ACLs; their values are not retained with the message and so cannot be used at delivery time. `$spam_score`

The spam score of the message, for example `“3.4”`; or `“30.5”`. This is useful for inclusion in log or reject messages. `$spam_score_int`

The spam score of the message, multiplied by ten, as an integer value. For example `“34”`; or `“305”`. This is useful for numeric comparisons in conditions. This variable is special; its value is saved with the message, and written to Exim's spool file. This means that it can be used during the whole life of the message on your Exim system, in particular, in routers or transports during the later delivery phase. `$spam_bar`

A string consisting of a number of `“+”`; or `“-”`; characters, representing the integer part of the spam score value. A spam score of 4.4 would have a `$spam_bar` value of `“4”`. This is useful for inclusion in warning headers, since MUAs can match on such strings. `$spam_report`

A multiline text table, containing the full SpamAssassin report for the message. Useful for inclusion in headers or reject messages.

The spam condition caches its results. If you call it again with the same user name, it does not scan again, but rather returns the same values as before.

The spam condition returns DEFER if there is any error while running the message through SpamAssassin. If you want to treat DEFER as FAIL (to pass on to the next ACL statement block), append `/defer_ok` to the right-hand side of the spam condition, like this: `deny message = This message was classified as SPAM`

```
spam = joe/defer_ok
```

This causes messages to be accepted even if there is a problem with spamd.

Here is a longer, commented example of the use of the spam condition: `# put headers in all messages (no matter if spam or not)`

```
warn message = X-Spam-Score: $spam_score ($spam_bar)
spam = nobody:true
warn message = X-Spam-Report: $spam_report
spam = nobody:true
```

```
# add second subject line with *SPAM* marker when message
# is over threshold
```

```
warn message = Subject: *SPAM* $h_Subject:
spam = nobody
```

```
# reject spam at high scores (> 12)
```

```
deny message = This message scored $spam_score spam points.
spam = nobody:true
condition = ${if >{$spam_score_int}{120}{1}{0}}
```

40.4 Scanning MIME parts

The `acl_smtp_mime` global option specifies an ACL that is called once for each MIME part of an SMTP message, including multipart types, in the sequence of their position in the message. Similarly, the `acl_not_smtp_mime` option specifies an ACL that is used for the MIME parts of non-SMTP messages. These options may both refer to the same ACL if you want the same processing in both cases.

These ACLs are called (possibly many times) just before the `acl_smtp_data` ACL in the case of an SMTP message, or just before a non-SMTP message is accepted. However, a MIME ACL is called only if the message contains a `MIME-Version:` header line. When a call to a MIME ACL does not yield `“accept”`, ACL processing is aborted and the appropriate result code is sent to the client. In the case of an SMTP message, the `acl_smtp_data` ACL is not called when this happens.

You cannot use the malware or spam conditions in a MIME ACL; these can only be used in the DATA or non-SMTP ACLs. However, you can use the regex condition to match against the raw MIME part. You can also use the `mime_regex` condition to match against the decoded MIME part (see section 40.5).

At the start of a MIME ACL, a number of variables are set from the header information for the relevant MIME part. These are described below. The contents of the MIME part are not by default decoded into a disk file except for MIME parts

whose content-type is `“message/rfc822”`. If you want to decode a MIME part into a disk file, you can use the decode modifier. The general syntax is: `decode = [/<path>/]<filename>`

The right hand side is expanded before use. After expansion, the value can be:

-

`“0”`; or `“false”`;, in which case no decoding is done.

-

The string `“default”`. In that case, the file is put in the temporary `“default”` directory `<spool_directory>/scan/<message_id>/` with a sequential file name consisting of the message id and a sequence number. The full path and name is available in `$mime_decoded_filename` after decoding.

-

A full path name starting with a slash. If the full name is an existing directory, it is used as a replacement for the default directory. The filename is then sequentially assigned. If the path does not exist, it is used as the full path and file name.

-

If the string does not start with a slash, it is used as the filename, and the default path is then used.

You can easily decode a file with its original, proposed filename using `decode = $mime_filename`

However, you should keep in mind that `$mime_filename` might contain anything. If you place files outside of the default path, they are not automatically unlinked.

For RFC822 attachments (these are messages attached to messages, with a content-type of `“message/rfc822”`), the ACL is called again in the same manner as for the primary message, only that the `$mime_is_rfc822` expansion variable is set (see below). Attached messages are always decoded to disk before being checked, and the files are unlinked once the check is done.

The MIME ACL supports the `regex` and `mime_regex` conditions. These can be used to match regular expressions against raw and decoded MIME parts, respectively. They are described in section 40.5.

The following list describes all expansion variables that are available in the MIME ACL: `$mime_boundary`

If the current part is a multipart (see `$mime_is_multipart`) below, it should have a boundary string, which is stored in this variable. If the current part has no boundary parameter in the Content-Type: header, this variable contains the empty string. `$mime_charset`

This variable contains the character set identifier, if one was found in the Content-Type: header. Examples for charset identifiers are: `us-ascii`
`gb2312 (Chinese)`
`iso-8859-1`

Please note that this value is not normalized, so you should do matches case-insensitively. `$mime_content_description`

This variable contains the normalized content of the Content-Description: header. It can contain a human-readable description of the parts content. Some implementations repeat the filename for attachments here, but they are usually only used for display purposes. `$mime_content_disposition`

This variable contains the normalized content of the Content-Disposition: header. You can expect strings like `“attachment”`; or `“inline”`; here. `$mime_content_id`

This variable contains the normalized content of the Content-ID: header. This is a unique ID that can be used to reference a part from another part. `$mime_content_size`

This variable is set only after the decode modifier (see above) has been successfully run. It contains the size of the decoded part in kilobytes. The size is always rounded up to full kilobytes, so only a completely empty part has a `$mime_content_size` of zero. `$mime_content_transfer_encoding`

This variable contains the normalized content of the Content-transfer-encoding: header. This is a symbolic name for an encoding type. Typical values are `“base64”`; and `“quoted-printable”`; `$mime_content_type`

If the MIME part has a Content-Type: header, this variable contains its value, lowercased, and without any options (like “name” or “charset”). Here are some examples of popular MIME types, as they may appear in this variable:

```
text/plain
text/html
application/octet-stream
image/jpeg
audio/midi
```

If the MIME part has no Content-Type: header, this variable contains the empty string. `$mime_decoded_filename`

This variable is set only after the decode modifier (see above) has been successfully run. It contains the full path and file name of the file containing the decoded data.

`$mime_filename`

This is perhaps the most important of the MIME variables. It contains a proposed filename for an attachment, if one was found in either the Content-Type: or Content-Disposition: headers. The filename will be RFC2047 decoded, but no additional sanity checks are done. If no filename was found, this variable contains the empty string. `$mime_is_coverletter`

This variable attempts to differentiate the “cover letter” of an e-mail from attached data. It can be used to clamp down on flashy or unnecessarily encoded content in the cover letter, while not restricting attachments at all.

The variable contains 1 (true) for a MIME part believed to be part of the cover letter, and 0 (false) for an attachment. At present, the algorithm is as follows:

-

The outermost MIME part of a message is always a cover letter.

-

If a multipart/alternative or multipart/related MIME part is a cover letter, so are all MIME subparts within that multipart.

-

If any other multipart is a cover letter, the first subpart is a cover letter, and the rest are attachments.

-

All parts contained within an attachment multipart are attachments.

As an example, the following will ban “HTML mail” (including that sent with alternative plain text), while allowing HTML files to be attached. HTML coverletter mail attached to non-HMTL coverletter mail will also be allowed: deny message = HTML mail is not accepted here

```
!condition = $mime_is_rfc822
condition = $mime_is_coverletter
condition = ${if eq{$mime_content_type}{text/html}{1}{0}}
$mime_is_multipart
```

This variable has the value 1 (true) when the current part has the main type “multipart”, for example “multipart/alternative” or “multipart/mixed”. Since multipart entities only serve as containers for other parts, you may not want to carry out specific actions on them. `$mime_is_rfc822`

This variable has the value 1 (true) if the current part is not a part of the checked message itself, but part of an attached message. Attached message decoding is fully recursive. `$mime_part_count`

This variable is a counter that is raised for each processed MIME part. It starts at zero for the very first part (which is usually a multipart). The counter is per-message, so it is reset when processing RFC822 attachments (see `$mime_is_rfc822`). The counter stays set after `acl_smtp_mime` is complete, so you can use it in the DATA ACL to determine the number of MIME parts of a message. For non-MIME messages, this variable contains the value -1.

40.5 Scanning with regular expressions

You can specify your own custom regular expression matches on the full body of the message, or on individual MIME parts.

The regex condition takes one or more regular expressions as arguments and matches them against the full message (when called in the DATA ACL) or a raw MIME part (when called in the MIME ACL). The regex condition matches

linewise, with a maximum line length of 32K characters. That means you cannot have multiline matches with the regex condition.

The mime_regex condition can be called only in the MIME ACL. It matches up to 32K of decoded content (the whole content at once, not linewise). If the part has not been decoded with the decode modifier earlier in the ACL, it is decoded automatically when mime_regex is executed (using default path and filename values). If the decoded data is larger than 32K, only the first 32K characters are checked.

The regular expressions are passed as a colon-separated list. To include a literal colon, you must double it. Since the whole right-hand side string is expanded before being used, you must also escape dollar signs and backslashes with more backslashes, or use the \N facility to disable expansion. Here is a simple example that contains two regular expressions: deny message = contains blacklisted regex (\$regex_match_string)
 regex = [Mm]ortgage : URGENT BUSINESS PROPOSAL

The conditions returns true if any one of the regular expressions matches. The \$regex_match_string expansion variable is then set up and contains the matching regular expression.

Warning: With large messages, these conditions can be fairly CPU-intensive. 40.6 The demime condition

The demime ACL condition provides MIME unpacking, sanity checking and file extension blocking. It is usable only in the DATA and non-SMTP ACLs. The demime condition uses a simpler interface to MIME decoding than the MIME ACL functionality, but provides no additional facilities. Please note that this condition is deprecated and kept only for backward compatibility. You must set the WITH_OLD_DEMIME option in Local/Makefile at build time to be able to use the demime condition.

The demime condition unpacks MIME containers in the message. It detects errors in MIME containers and can match file extensions found in the message against a list. Using this facility produces files containing the unpacked MIME parts of the message in the temporary scan directory. If you do antivirus scanning, it is recommended that you use the demime condition before the antivirus (malware) condition.

On the right-hand side of the demime condition you can pass a colon-separated list of file extensions that it should match against. For example: deny message = Found blacklisted file attachment
 demime = vbs:com:bat:pif:prf:lnk

If one of the file extensions is found, the condition is true, otherwise it is false. If there is a temporary error while demimeing (for example, "disk full"), the condition defers, and the message is temporarily rejected (unless the condition is on a warn verb).

The right-hand side is expanded before being treated as a list, so you can have conditions and lookups there. If it expands to an empty string, "false", or zero ("0"), no demimeing is done and the condition is false.

The demime condition set the following variables: \$demime_errorlevel

When an error is detected in a MIME container, this variable contains the severity of the error, as an integer number. The higher the value, the more severe the error (the current maximum value is 3). If this variable is unset or zero, no error occurred. \$demime_reason

When \$demime_errorlevel is greater than zero, this variable contains a human-readable text string describing the MIME error that occurred. \$found_extension

When the demime condition is true, this variable contains the file extension it found.

Both \$demime_errorlevel and \$demime_reason are set by the first call of the demime condition, and are not changed on subsequent calls.

If you do not want to check for file extensions, but rather use the demime condition for unpacking or error checking purposes, pass "*" as the right-hand side value. Here is a more elaborate example of how to use this facility: # Reject messages with serious MIME container errors
 deny message = Found MIME error (\$demime_reason).
 demime = *
 condition = \${if >{\$demime_errorlevel}{2}{1}{0}}

```
# Reject known virus spreading file extensions.  
# Accepting these is pretty much braindead.  
deny message = contains $found_extension file (blacklisted).  
  demime = com:vbs:bat:pif:scr  
  
# Freeze .exe and .doc files. Postmaster can  
# examine them and eventually thaw them.  
deny log_message = Another $found_extension file.  
  demime = exe:doc  
  control = freeze
```