S. Bosch

July 7, 2016

Sieve Email Filtering: Invoking External Programs
spec-bosch-sieve-pipe

Abstract

The Sieve filtering language (RFC 5228) is explicitly designed to be powerful enough to be useful yet limited in order to allow for a safe filtering system. The base specification of the language makes it impossible for users to do anything more complex (and dangerous) than write simple mail filters. One of the consequences of this security-minded design is that users cannot execute programs external to the Sieve filter. However, this can be a very useful and flexible feature for situations where Sieve cannot provide some uncommon functionality by itself. This document updates the Sieve filtering language with extensions that add support for invoking a predefined set of external programs. Messages can be piped to or filtered through those programs and string data can be input to and retrieved from those programs.

Table of Contents

☐

1. Introduction

This is an extension to the Sieve filtering language defined by RFC 5228 [SIEVE]. It adds commands for invoking a predefined set of external programs. Messages can be piped to or filtered through those programs and, alternatively, string data can be passed to and retrieved from those programs.

The Sieve language is explicitly designed to be powerful enough to be useful yet limited in order to allow for a safe server-side filtering system. Therefore, the base specification of the language makes it impossible for users to do anything more complex (and dangerous) than write simple mail filters. One of the consequences of this security-minded design is that users cannot execute external programs from their Sieve script. Particularly for server-side filtering setups in which mail accounts have no corresponding system account, allowing the execution of arbitrary programs from the mail filter can be a significant security risk. However, such functionality can also be very useful, for instance to easily implement a custom action or external effect that Sieve normally cannot provide.

This document updates the Sieve filtering language with an extension to support invoking a predefined set of external programs using a set of new commands. To mitigate the security concerns, the external programs cannot be chosen arbitrarily; the available programs are restricted through administrator configuration.

This extension is specific to the Pigeonhole Sieve implementation for the Dovecot Secure IMAP server. It will therefore most likely not be supported by web interfaces and GUI-based Sieve editors. This extension is primarily meant for use in small setups or global scripts that are managed by the system's administrator.

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [KEYWORDS].

Conventions for notations are as in [SIEVE] Section 1.1, including use of the "Usage:" label for the definition of action and tagged arguments syntax.

3. Naming of External Programs

An external program is identified by a name. This MUST not correspond to a file system path or otherwise have the ability to point to arbitrary programs on the system. The list of valid program names MUST be limited, subject to administrator configuration.

A program name is a sequence of Unicode characters encoded in UTF-8 [UTF-8]. A program name MUST comply with Net-Unicode Definition (Section 2 of [NET-UNICODE]), with the additional restriction of prohibiting the following Unicode characters:

- o 0000-001F; [CONTROL CHARACTERS]
- o 002F; SLASH
- o 007F; DELETE
- o 0080-009F; [CONTROL CHARACTERS]
- o 2028; LINE SEPARATOR
- o 2029; PARAGRAPH SEPARATOR

Program names MUST be at least one octet (and hence Unicode character) long. Implementations MUST allow names of up to 128 Unicode characters in length (which can take up to 512 octets when encoded in UTF-8, not counting the terminating NUL), and MAY allow longer names. A server that receives a program name longer than its internal limit MUST reject the corresponding operation, in particular it MUST NOT truncate the program name.

Implementations MUST NOT allow variables to be expanded into the program names; in other words, the "program-name" value MUST be a constant string as defined in [VARIABLES], Section 3.

4. Arguments for External Programs

Optionally, arguments can be passed to an external program. The arguments are specified as a Sieve string list and are passed to the external program in sequence. Implementations SHOULD NOT impose any structure for these arguments; validity checks are the responsibility of the external program.

However, implementations SHOULD limit the maximum number of arguments and the length of each argument. Implementations MUST accept at least 16 arguments with a length of at least 1024 octets each, and MAY allow more and longer arguments. Additionally, implementations MAY restrict the use of certain control characters such as CR and LF, if these can cause unexpected behavior or raise security concerns.

□

Note that implementations MAY also implicitly pass other data, such as the message envelope, to all executed programs avoiding the need to pass this information explicitly through program arguments.

5. Action "pipe"

```
   Usage: "pipe" [":try"] <program-name: string>
                 [<arguments: string-list>]
```

The "pipe" action executes the external program identified by the "program-name" argument and pipes the message to it. Much like the "fileinto" and "redirect" actions [SIEVE], this action is a disposition-type action (it is intended to deliver the message) and therefore it cancels Sieve's implicit keep (see Section 2.10.2 of [SIEVE]) by default.

The specified "program-name" argument MUST conform to the syntax and restrictions defined in Section 3. A script MUST fail with an appropriate error if it attempts to use the "filter" action with an invalid, restricted or unknown program name. The optional "arguments" argument lists the arguments that are passed to the external program, as explained in Section 4.

If the external program invoked by the "pipe" action fails to execute or finishes execution with an error, script execution MUST fail with an appropriate error (causing an implicit "keep" action to be executed), unless the ":try" tag is specified.

When the ":try" tag is specified, the "pipe" instruction will attempt execution of the external program, but failure will not cause the whole Sieve script execution to fail with an error. Instead, the Sieve processing continues as if the "pipe" action was never triggered.

If the execution of the external program is unsuccessful, the "pipe" action MUST NOT cancel the implicit keep.

5.1. Interactions with Other Sieve Actions

By default, the "pipe" action cancels the implicit keep, thereby handing the responsibility for the message over to the external program. This behavior can be overridden using the Sieve "copy" extension [RFC3894] as described in Section 5.2.

The "pipe" action can only be executed once per script for a particular external program. A script MUST fail with an appropriate error if it attempts to "pipe" messages to the same program multiple times.

The "pipe" action is incompatible with the Sieve "reject" and "ereject" actions [RFC5429].

5.2. Interaction with the Sieve "copy" Extension

The Sieve "copy" extension [RFC3894] adds an optional ":copy" tagged argument to the "fileinto" and "redirect" action commands. When this tag is specified, these commands do not cancel the implicit "keep". Instead, the requested action is performed in addition to whatever else is happening to the message.

When the "vnd.dovecot.pipe" extension is active, the "copy" extension also adds the optional ":copy" tag to the "pipe" action command. This has the familiar effect that when the ":copy" tag is specified, the implicit "keep" will not be canceled by the "pipe" action. When the "copy" extension is active, the syntax of the "pipe" action is represented as follows:

```
Usage: "pipe" [":copy"] [":try"] <program-name: string>
              [<arguments: string-list>]
```

6. Action "filter"

```
Usage: "filter" <program-name: string> [<arguments: string-list>]
```

The "filter" action executes the external program identified by the "program-name" argument and filters the message through it. This means that the message is provided as input to the external program and that the output of the external program is used as the new message. This way, the entire message can be altered using the external program. The "filter" action does not affect Sieve's implicit keep.

The specified "program-name" argument MUST conform to the syntax and restrictions defined in Section 3. A script MUST fail with an appropriate error if it attempts to use the "filter" action with an invalid, restricted or unknown program name. The optional "arguments" argument lists the arguments that are passed to the external program, as explained in Section 4.

If the external program fails to execute, finishes execution with an error, or fails to provide message output, the "filter" action MUST terminate and leave the message unchanged. Depending on the severity of the error, implementations MAY subsequently fail the entire script execution with an appropriate error (causing an implicit "keep" action to be executed). If no error condition is raised, script processing continues, and prior or subsequent "filter" actions are not affected.

6.1. Interaction with Other Tests and Actions

A successful "filter" action effectively changes the message, potentially substituting the message in its entirety with a new version. However, actions such as "reject" and "vacation" that generate [MDN], [DSN], or similar disposition messages MUST do so using the original, unmodified message. Similarly, if an error terminates processing of the script, the original message MUST be used when doing the implicit keep required by Section 2.10.6 of [SIEVE]. All other actions that store, send, or alter the message MUST do so with the current version of the message. This includes the "filter" action itself.

When a disposition-type action, such as "fileinto", "redirect" or "pipe", is encountered, the current version of the message is "locked in" for that disposition-type action. Whether the implementation performs the action at that point or batches it for later, it MUST perform the action on the message as it stood at the time, and MUST NOT include subsequent changes encountered later in the script processing.

In addition, any tests done on the message and its parts will test the message after all prior "filter" actions have been performed. Because the implicit keep, if it is in effect, acts on the final state of the message, all "filter" actions are performed before any implicit keep.

The "filter" action does not affect the applicability of other actions; any action that was applicable before the "filter" invocation is equally applicable to the changed message afterward.

7. Action "execute"

```
Usage: "execute" [":input" <input-data: string> / ":pipe"]
                 [":output" <varname: string>]
                 <program-name: string> [<arguments: string-list>]
```

The "execute" action executes the external program identified by the "program-name" argument. Input to the program can be provided using the ":input" or ":pipe" tags. If used in combination with the "variables" extension [VARIABLES], the "execute" action can redirect output from the program to the variable specified using the ":output" tag. This way, string data can be passed to and retrieved from an external program. The "execute" action does not change the message in any way and it never affects Sieve's implicit keep.

The specified "program-name" argument MUST conform to the syntax and restrictions defined in Section 3. A script MUST fail with an appropriate error if it attempts to use the "execute" action with an invalid, restricted or unknown program name. The optional "arguments" argument lists the arguments that are passed to the external program, as explained in Section 4.

The ":input" and ":pipe" tags are mutually exclusive, because these both specify input that is passed to the external program. Specifying both for a single "execute" command MUST trigger a compile error. The ":input" tag specifies a string that is passed to the external script as input. This string may also contain variable substitutions when the "variables" extension is active. If instead the ":pipe" tag is specified, the current version of the message itself is passed to the external program. If the ":input" and ":pipe" tags are both omitted, no input is provided to the external program.

The ":output" tag specifies the variable to which the output of the external program is to be redirected. If the ":output" tag is omitted, any output from the external program is discarded. The ":output" tag requires the "variables" [VARIABLES] extension to be active. The use of the ":output" tag for the "execute" action without the "variables" extension in the require line MUST trigger a compile error.

The "varname" parameter of the ":output" tag specifies the name of the variable. It MUST be a constant string and it MUST conform to the syntax of "variable-name" as defined in [VARIABLES], Section 3. An invalid name MUST be detected as a syntax error. The referenced variable MUST be compatible with the "set" command as described in [VARIABLES], Section 4. This means that match variables cannot be specified and that variable namespaces are only allowed when their specification explicitly indicates compatibility with the "set" command. Use of an incompatible variable MUST trigger a compile error. The data actually stored in the variable MAY be truncated to conform to an implementation-specific limit on variable length.

If the external program fails to execute or finishes execution with an error, the "execute" action MUST terminate and leave the contents of the variable referenced with ":output" unchanged. Depending on the severity of the error, implementations MAY subsequently fail the entire script execution with an appropriate error (causing an implicit "keep" action to be executed).

8. Actions "filter" and "execute" as Tests

To simplify checking the successful invocation of the external program, the "filter" and "execute" actions can also be used as tests. As such, these will attempt to execute the requested external program, and will evaluate to "true" if the program executed successfully and, if applicable, output was retrieved from it successfully. The usage as a test is exactly the same as the usage as an action: as a test it doubles as an action and a test of the action's result at the same time.

For the "execute" test, a "false" result is not necessarily equal to actual failure: it may just mean that the executed program returned a "false" result, e.g. an exit code higher than zero on Unix systems. Note that any output from the external program is discarded when it yields a "false" result. Similarly, for the "filter" test, programs may return a "false" result to indicate that the message was not changed. In that case the Sieve interpreter will not replace the active message with an identical one, which is beneficial for efficiency. The exact semantics of these tests thus partly depends on the program being executed.

To handle missing programs gracefully, implementations MAY let the "filter" and "execute" tests evaluate to "false" if an unknown program name is specified, instead of failing the script with an error as would happen if used as an action. In any other case and irrespective of whether the command is used as an action or a test, passing invalid arguments to the "filter" or "execute" commands, such as a syntactically invalid or restricted program name, MUST always cause the script to fail with an appropriate error.

9. Sieve Capability Strings

A Sieve implementation that defines the "pipe" action command will advertise the capability string "vnd.dovecot.pipe".

A Sieve implementation that defines the "filter" action command will advertise the capability string "vnd.dovecot.filter".

A Sieve implementation that defines the "execute" command will advertise the capability string "vnd.dovecot.execute".

10. Examples

The examples outlined in this section all refer to some external program. These programs are imaginary and are only available when the administrator would provide them.

10.1. Example 1

The following example passes messages directed to a "user-request@example.com" address to an external program called "request-handler". The "-request" part of the recipient address is identified

□

using the "subaddress" extension [SUBADDRESS]. If the program is executed successfully, the message is considered delivered and does not end up in the user's inbox.

```
require [ "vnd.dovecot.pipe", "subaddress", "envelope" ];

if envelope :detail "to" "request"
{
  pipe "request-handler";
}
```

10.2. Example 2

The following example copies messages addressed to a particular recipient to a program called "printer". This program sends the message to some printer. In this case it is configured for "A4" page format and "draft" quality using the two arguments. Irrespective of whether the message is printed or not, it is also always stored in the user's inbox through Sieve's implicit keep action (which is not canceled due to the specified :copy tag).

```
require [ "vnd.dovecot.pipe", "copy" ];

if address "to" "snailmail@example.com"
{
  pipe :copy "printer" ["A4", "draft"];
}
```

10.3. Example 3

The following example translates a message from Dutch to English if appropriate. If the message's content language is indicated to be Dutch, the message is filtered through an external program called "translator" with arguments that request Dutch to English translation. Dutch messages are translated and filed into a special folder called "Translated". Other messages are delivered to the user's inbox.

```
require [ "vnd.dovecot.filter", "fileinto" ];

if header "content-language" "nl"
{
  filter "translator" ["nl", "en"];
  fileinto "Translated";
  stop;
}
```

Note that (formerly) Dutch messages are filed into the "Translated" folder, even when the "translator" program fails. In the following modified example this is prevented by using the filter action as a test:

```
require [ "vnd.dovecot.filter", "fileinto" ];

if header "content-language" "nl"
{
  if filter "translator" ["nl", "en"]
  {
    fileinto "Translated";
    stop;
  }
}
```

This way, messages only end up in the "Translated" folder when translation was actually successful.

10.4. Example 4

The following example determines whether the user is on vacation by querying an external source. The vacation message is obtained from the external source as well. The program that queries the external source is called "onvacation" and it has one argument: the localpart of the recipient address. The execute action is used as a test, which will evaluate to "true" when the user is determined to be on vacation. This means that the external program "onvacation" exits with a failure when the user is not on vacation. Of course, a vacation response is also not sent when the "onvacation" program truly fails somehow.

```
require [ "vnd.dovecot.execute", "vacation", "variables",
          "envelope" ];

if envelope :localpart :matches "to" "*"
{
  set "recipient" "${1}";
}

if execute :output "vacation_message" "onvacation" "${recipient}"
{
  vacation "${vacation_message}";
}
```

11. Security Considerations

Allowing users to execute programs external to the Sieve filter can be a significant security risk, therefore the extensions presented in this specification must be implemented with great care. The external programs should execute with no more privileges than needed.

Particularly the arguments passed to the external programs (see Section 4) need to be handled with scrutiny. The external programs need to check the arguments for validity and SHOULD NOT pass these to system tools directly, as this may introduce the possibility of various kinds of insertion attacks. External programs that work with message content or string input from the Sieve script may have similar security concerns.

Unlike the Sieve interpreter itself, an external program can easily consume a large amount of resources if not implemented carefully. This can be triggered by coincidence or intentionally by an attacker. Therefore, the amount of resources available to the external programs SHOULD be limited appropriately. For one, external programs MUST NOT be allowed to execute indefinitely.

For improved security, implementations MAY restrict the use of this extension to administrator-controlled global Sieve scripts. In such setups, the external programs are never called directly from the user's personal script. For example, using the "include" extension [INCLUDE], the user's personal script can include global scripts that contain the actual external program invocations. This both abstracts the details of external program invocation from the user's view and it limits access to external programs to whatever the administrator defines.

12. References

12.1. Normative References

[KEYWORDS]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[NET-UNICODE]
Klensin, J. and M. Padlipsky, "Unicode Format for Network Interchange", RFC 5198, March 2008.

[RFC3894] Degener, J., "Sieve Extension: Copying Without Side Effects", RFC 3894, October 2004.

[SIEVE]   Guenther, P. and T. Showalter, "Sieve: An Email Filtering Language", RFC 5228, January 2008.

[UTF-8]   Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[VARIABLES]
Homme, K., "Sieve Email Filtering: Variables Extension", RFC 5229, January 2008.

12.2. Informative References

[DSN]   Moore, K. and G. Vaudreuil, "An Extensible Message Format for Delivery Status Notifications", RFC 3464, January 2003.

[INCLUDE] Daboo, C. and A. Stone, "Sieve Email Filtering: Include Extension", RFC 6609, May 2012.

[MDN]   Hansen, T. and G. Vaudreuil, "Message Disposition Notification", RFC 3798, May 2004.

[RFC5429] Stone, A., "Sieve Email Filtering: Reject and Extended Reject Extensions", RFC 5429, March 2009.

[SUBADDRESS]
Murchison, K., "Sieve Email Filtering -- Subaddress Extension", RFC 3598, September 2003.

Author's Address

Stephan Bosch
Enschede
NL

Email: stephan@rename-it.nl