Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: September 19, 2007

T. Hansen
AT&T Laboratories
C. Daboo
Apple Computer

March 18, 2007

SIEVE Email Filtering: MIME part Tests, Iteration, Replacement and Enclosure
draft-ietf-sieve-mime-loop-02.txt

Abstract

The SIEVE email filtering language has no way to examine individual MIME parts or any way to manipulate those individual parts. However, being able to filter based on MIME content is important. This document defines extensions for these needs.

Note

This document is being discussed on the MTA-FILTERS mailing list, ietf-mta-filters@imc.org.

1. Introduction

SIEVE scripts are used to make decisions about the disposition of an email message. The base SIEVE specification, [I-D.ietf-sieve-3028bis], defines operators for looking at the message headers, such as addresses and the subject. Other extensions provide access to the body of the message ([I-D.ietf-sieve-body]), or allow you to manipulate the header of the message ([I-D.ietf-sieve-editheader]). But none of these extensions take into account that MIME messages ([RFC2045]) are often complex objects, consisting of many parts and sub-parts. This extension defines mechanisms for performing tests on MIME body parts, looping through the MIME body parts, changing the contents of a MIME body part, and enclosing the message with a wrapper.

2. Conventions Used in This Document

Conventions for notations are as in [I-D.ietf-sieve-3028bis] section 1.1.

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 [RFC2119].

3. SIEVE Loops

The base SIEVE language has no looping mechanism. Given that messages may contain multiple parts, in order to support filters that apply to any and all parts, we introduce a new control command: "for_every_part", which is an iterator that walks though every MIME part of a message, including nested parts, and applies the commands in the specified block to each of them. The iterator will start with the first MIME part (as its current context) and will execute a command block (SIEVE commands enclosed by { ...}). Upon completion of this command block, the iterator advances to the next MIME part (as its current context) and executes the same command block again.

The iterator can be terminated prematurely by a new SIEVE command, "break".

Usage: for_every_part block

Usage: break;

"for_every_part" commands can be nested inside other "for_every_part" commands. When this occurs, the nested "for_every_part" iterates over the MIME parts contained within the MIME part current being targeted by the nearest enclosing "for_every_part" command. If that MIME part is a terminal MIME part (i.e. does not contain other MIME parts) then the nested "for_every_loop" is simply ignored.

SIEVE implementations MAY limit the number of nested loops that occur within one another, however they MUST support at least one nested loop inside another loop.

4. Changes to SIEVE tests

This specification extends the base SIEVE "header", "address" and "exists" tests to support targeting those tests at a specific MIME part or at all MIME parts in the enclosing scope.

4.1. Test "header"

The "header" test is extended with the addition of a new ":mime" tagged argument, which takes a number of other arguments.

Usage: header [:mime] [:anychild] [MIMEOPTS]
[COMPARATOR] [MATCH-TYPE]
<header-names: string-list> <key-list: string-list>

Usage: The definition of [MIMEOPTS] is:

Syntax: ":type" / ":subtype" / ":contenttype" /
":param" <param-list: string-list>

When the ":mime" tagged argument is present in the "header" test, it will parse the MIME header lines in a message so that tests can be performed on specific elements.

If the ":anychild" tagged argument is NOT specified:

- If used within the context of a "for_every_part" iterator, the "header" test will examine the headers associated with the current MIME part context from the loop.

- If used outside the context of a "for_every_part" iterator, the "header" test will examine only the outer, top-level, headers of

the message.

If the ":anychild" tagged argument IS specified, the "header" test will examine all MIME body parts and return true if any of them satisfies the test.

The "header" test with the ":mime" tagged argument can test various aspects of certain structed MIME headers. These options are available:

:type parses the header assuming it has the format of a "Content-Type:" MIME header field, and tests the value of the MIME type specified in the header.

:subtype parses the header assuming it has the format of a "Content-Type:" MIME header field, and tests the value of the MIME subtype specified in the header.

:contenttype parses the header assuming it has the format of a "Content-Type:" MIME header field, and tests the combined value of the MIME type and subtype specified in the header.

:param parses the header looking for MIME parameters in the header. The supplied string-list lists the names of any parameters to be tested. If any one named parameter value matches the test string value, the test will return true.

Example:

```
require ["mime", "fileinto"];

if header :mime :type "Content-Type" "image"
{
    fileinto "INBOX.images";
}
```

In this example, any message that contains a MIME image type part at the top-level is saved to the mailbox "INBOX.images".

Example:

```
require ["mime", "fileinto"];

if header :mime :anychild :contenttype :comparator
          "Content-Type" "text/html"
{
    fileinto "INBOX.html";
}
```

In this example, any message that contains any MIME part with a content-type of "text/html" is saved to the mailbox "INBOX.html".

Example:

```
require ["mime", "for_every_part", "fileinto"];

for_every_part
{
    if header :mime :param "filename" :comparator
       "Content-Disposition" "important"
    {
        fileinto "INBOX.important";
        break;
    }
}
```

In this example, any message that contains any MIME part with a content-disposition with a filename parameter containing the text "important" is saved to the mailbox "INBOX.important".

4.2. Test "address"

The "address" test is extended with the addition of a new ":mime" tagged argument, which takes a number of other arguments.

```
Usage:  address [:mime] [:anychild] [COMPARATOR]
   [ADDRESS-PART] [MATCH-TYPE]
   <header-list: string-list> <key-list: string-list>
```

When the ":mime" tagged argument is present in the "address" test, it will parse the MIME header lines as if they were standard address header lines in a message so that tests can be performed on specific elements.

The behavior of the ":anychild" tagged argument and the interaction with the "for_every_part" iterator is the same as for the extended "header" test Section 4.1.

Example:

```
require ["mime", "fileinto"];

if address :mime :is :all "content-from" "tim@example.com"
{
    fileinto "INBOX.part-from-tim";
}
```

In this example, any message that contains a MIME Content-From header at the top-level matching the text "tim@example.com" is saved to the mailbox "INBOX.part-from-time".

4.3. Test "exists"

The "exists" test is extended with the addition of a new ":mime" tagged argument, which takes one other argument.

Usage: exists [:mime] [:anychild] <header-names: string-list>

When the ":mime" tagged argument is present in the "exists" test, the test is extended to check for the existence of MIME headers in MIME parts.

The behavior of the ":anychild" tagged argument and the interaction with the "for_every_part" iterator is the same as for the extended "header" test Section 4.1.

Example:

```
require ["mime", "fileinto"];

if exists :mime :anychild "content-md5"
{
    fileinto "INBOX.md5";
}
```

In this example, any message that contains a MIME Content-MD5 header in any MIME part is saved to the mailbox "INBOX.md5".

5. Action Replace

Usage: replace [:mime] [:subject string] [:from string]
<replacement: string>

The "replace" command is defined to allow a MIME part to be replaced with the text supplied in the command.

When used in the context of a "for_every_part" iterator, the MIME part to be replaced is the "current" MIME part. If the current MIME context is a multipart MIME part, the entire multipart MIME part is replaced, which would alter the MIME structure of the message by eliminating all of the children of the multipart part. (Replacing a non-multipart MIME part within a "for_every_part" loop context does not alter the overall message structure.) If the MIME structure is altered, the change takes effect immediately: the "for_every_part"

iterator that is executing does not go into the no-longer existing body parts, and subsequent "for_every_part" iterators would use the new message structure.

When used outside the context of a "for_every_part" loop, the MIME part to be replaced is the entire message.

If the :mime parameter is not specified, the replacement string is a text/plain part.

If the :mime parameter is specified, then the replacement string is, in fact, a MIME entity as defined in [RFC2045] section 2.4, including both MIME headers and content. If the optional :mime parameter is not supplied, the reason string is considered to be a UTF-8 string.

If the entire message is being replaced, a ":subject" parameter specifies a subject line to attach to the message that is generated. UTF-8 characters can be used in the string argument; implementations MUST convert the string to [RFC2047] encoded words if and only if non-ASCII characters are present. Implementations MUST preserve the previous Subject header as an Original-Subject header.

If the entire message is being replaced, a ":from" parameter may be used to specify an alternate address to use in the From field of the message that is generated. The string must specify a valid [RFC2822] mailbox-list. Implementations SHOULD check the syntax and generate an error when a syntactically invalid ":from" parameter is specified. Implementations MAY also impose restrictions on what addresses can be specified in a ":from" parameter; it is suggested that values that fail such a validity check simply be ignored rather than causing the replace action to fail. Implementations MUST preserve the previous From header as an Original-From header.

6. Action Enclose

Usage: enclose <:subject string> <:headers string-list> string

A new SIEVE action command is defined to allow an entire message to be enclosed as an attachment to a new message. NB: The following statement may be controversial: This enclose action takes precedence over all other message modifications, such as "replace". If multiple "enclose" actions are executed by a script, only the text specified on the last one is used when creating the enclosed message. This action does not affect messages that are forwarded via a "redirect" action.

Specifically, the original message becomes a multipart/mixed message

with two parts: a text/plain portion with the string argument as its body, and a message/rfc822 portion with the original message enclosed. The Content-Type: header field becomes multipart/mixed. The Subject: header is specified by the :subject argument. Any headers specified by :headers are copied from the old message into the new message. NB: The following statement may be controversial: If not specified by :headers, Date: and From: headers should be synthesized to reflect the current date and the user running the SIEVE action.

7. SIEVE Capability Strings

A SIEVE implementation that defines the "for_every_part" and "break" actions will advertise the capability string "for_every_part".

A SIEVE implementation that defines the ":mime" tagged arguments to the "header", "address" and "exists" commands will advertise the capability string "mime".

A SIEVE implementation that defines the "replace" action will advertise the capability string "replace".

A SIEVE implementation that defines the "enclose" action will advertise the capability string "enclose".

8. Examples

8.1. Example 1

A SIEVE script to replace all the Windows executable attachments in a message would be:

```
require [ "for_every_part", "mime", "replace" ];
for_every_part
{
  if ( anyof (
         header :mime :contenttype :is "Content-Type" "application/exe",
         header :mime :param "filename"
           ["Content-Type", "Content-Disposition"] :matches "*.com" )
  {
    replace "Executable attachment removed by user filter";
  }
}
```

8.2. Example 2

A SIEVE script to warn the user about executable attachment types would be:

```
require [ "for_every_part", "mime", "enclose" ];

for_every_part
{
  if header :mime :param "filename"
     ["Content-Type", "Content-Disposition"] :matches
       ["*.com", "*.exe", "*.vbs", "*.scr",
        "*.pif", "*.hta", "*.bat", "*.zip" ]
  {
    # these attachment types are executable
    enclose :subject "Warning" "
WARNING! The enclosed message contains executable attachments.
These attachments types may contain a computer virus program
that can infect your computer and potentently damage your data

Before clicking on these message attachments, you should verify
with the sender that this message was sent by them and not a
computer virus.
";
    break;
  }
}
```

9. Acknowledgements

Comments from members of the MTA Filters Working Group, in particular Ned Freed, Nigel Swinson and Mark Mallett, are gratefully acknowledged.

10. Security Considerations

To be provided

11. IANA Considerations

The Original-Subject: and Original-From: headers are to be registered in the Permanent Message Header Fields table.

12. Change History (to be removed prior to publication as an RFC)

12.1. draft-ietf-sieve-mime-02

minor syntax glitches in examples

Add clarification on "replace" affecting subsequent for_every_part loops?

Add IANA considerations for Original-Subject: and Original-From:.

Add note on "enclose" creating From: and Date: headers.

12.2. draft-ietf-sieve-mime-01

what happens when nested for_every_loop's

a "mime" shorthand for testing the type/subtype, without requiring

interactions with variables
notifications
notifications to calendar service
address tests, exists tests
mimeheader, mimeparameter tests

12.3. draft-ietf-sieve-mime-00

Changed title and text to emphasize MIME Tests.

Changed for.every.part to for_every_part.

Added :anychild to mime test. Default is to use the current context or outer envelope; specifying :anychild will look at all children.

Added clarifications to replacing parts affecting the structure.

Added :mime option to replace, ala draft-ietf-sieve-vacation-06.

Various other minor nit fixes.

12.4. draft-hansen-sieve-loop-01

Merged with draft-daboo-sieve-mime-00.txt.

12.5. draft-hansen-sieve-loop-02

Update to 3028bis reference.

Added 2119 conventions section.

Terminology/title tweaks.

Added informative references to body and editheader extensions.

Added description of nested loops.

Replaced mime test by extensions to header, address and exists tests.

13. References

13.1. Normative References

[I-D.ietf-sieve-3028bis]
Showalter, T. and P. Guenther, "Sieve: An Email Filtering Language", draft-ietf-sieve-3028bis-12 (work in progress), February 2007.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2822] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

13.2. Informative References

[I-D.ietf-sieve-body]
Guenther, P. and J. Degener, "Sieve Email Filtering: Body Extension", draft-ietf-sieve-body-06 (work in progress), February 2007.

[I-D.ietf-sieve-editheader]
Guenther, P. and J. Degener, "Sieve Email Filtering: Editheader Extension", draft-ietf-sieve-editheader-08 (work in progress), March 2007.

Authors' Addresses

Tony Hansen
AT&T Laboratories
200 Laurel Ave.
Middletown, NJ 07748
USA

Email: tony+sieveloop@maillennium.att.com

Cyrus Daboo
Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
USA

Email: cyrus@daboo.name
URI: http://www.apple.com/

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).