Network Working Group
Request for Comments: 5260
Category: Standards Track

N. Freed
Sun Microsystems
July 2008

Sieve Email Filtering: Date and Index Extensions

Status of This Memo

Abstract

This document describes the "date" and "index" extensions to the Sieve email filtering language. The "date" extension gives Sieve the ability to test date and time values in various ways. The "index" extension provides a means to limit header and address tests to specific instances of header fields when header fields are repeated.

Table of Contents

1. Introduction 2
2. Conventions Used in This Document 2
3. Capability Identifiers 3
4. Date Test 3
  4.1. Zone and Originalzone Arguments 4
  4.2. Date-part Argument 4
  4.3. Comparator Interactions with Date-part Arguments 5
  4.4. Examples 6
5. Currentdate Test 6
  5.1. Examples 6
6. Index Extension 7
  6.1. Example 8
7. Security Considerations 8
8. IANA Considerations 9
9. References 9
  9.1. Normative References 9
  9.2. Informative References 10
Appendix A. Julian Date Conversions 11
Appendix B. Acknowledgements 12

1. Introduction

Sieve [RFC5228] is a language for filtering email messages at or around the time of final delivery. It is designed to be implementable on either a mail client or mail server. It is meant to be extensible, simple, and independent of access protocol, mail architecture, and operating system. It is suitable for running on a mail server where users may not be allowed to execute arbitrary programs, such as on black box Internet Message Access Protocol [RFC3501] servers, as it does not have user-controlled loops or the ability to run external programs.

The "date" extension provides a new date test to extract and match date/time information from structured header fields. The date test is similar in concept to the address test specified in [RFC5228], which performs similar operations on addresses in header fields.

The "date" extension also provides a currentdate test that operates on the date and time when the Sieve script is executed.

Some header fields containing date/time information, e.g., Received:, naturally occur more than once in a single header. In such cases it is useful to be able to restrict the date test to some subset of the fields that are present. For example, it may be useful to apply a date test to the last (earliest) Received: field. Additionally, it may also be useful to apply similar restrictions to either the header or address tests specified in [RFC5228].

For this reason, this specification also defines an "index" extension. This extension adds two additional tagged arguments :index and :last to the header, address, and date tests. If present, these arguments specify which occurrence of the named header field is to be tested.

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 RFC 2119 [RFC2119].

The terms used to describe the various components of the Sieve language are taken from Section 1.1 of [RFC5228]. Section 2 of the same document describes basic Sieve language syntax and semantics. The date-time syntactic element defined using ABNF notation [RFC5234] in [RFC3339] is also used here.

3. Capability Identifiers

The capability strings associated with the two extensions defined in this document are "date" and "index".

4. Date Test

```
Usage:   date [<":zone" <time-zone: string>> / ":originalzone"]
              [COMPARATOR] [MATCH-TYPE] <header-name: string>
              <date-part: string> <key-list: string-list>
```

The date test matches date/time information derived from headers containing [RFC2822] date-time values. The date/time information is extracted from the header, shifted to the specified time zone, and the value of the given date-part is determined. The test returns true if the resulting string matches any of the strings specified in the key-list, as controlled by the comparator and match keywords. The date test returns false unconditionally if the specified header field does not exist, the field exists but does not contain a syntactically valid date-time specification, the date-time isn’t valid according to the rules of the calendar system (e.g., January 32nd, February 29 in a non-leap year), or the resulting string fails to match any key-list value.

The type of match defaults to ":is" and the default comparator is "i;ascii-casemap".

Unlike the header and address tests, the date test can only be applied to a single header field at a time. If multiple header fields with the same name are present, only the first field that is found is used. (Note, however, that this behavior can be modified with the "index" extension defined below.) These restrictions simplify the test and keep the meaning clear.

The "relational" extension [RFC5231] adds a match type called ":count". The count of a date test is 1 if the specified field exists and contains a valid date; 0, otherwise.

Implementations MUST support extraction of RFC 2822 date-time information that either makes up the entire header field (e.g., as it does in a standard Date: header field) or appears at the end of a header field following a semicolon (e.g., as it does in a standard Received: header field). Implementations MAY support extraction of date and time information in RFC2822 or other formats that appears in other positions in header field content. In the case of a field containing more than one date or time value, the last one that appears SHOULD be used.

4.1. Zone and Originalzone Arguments

The :originalzone argument specifies that the time zone offset originally in the extracted date-time value should be retained. The :zone argument specifies a specific time zone offset that the date-time value is to be shifted to prior to testing. It is an error to specify both :zone and :originalzone.

The value of time-zone MUST be an offset relative to UTC with the following syntax:

```
    time-zone  =  ( "+" / "-" ) 4DIGIT
```

The "+" or "-" indicates whether the time-of-day is ahead of (i.e., east of) or behind (i.e., west of) UTC. The first two digits indicate the number of hours difference from Universal Time, and the last two digits indicate the number of minutes difference from Universal Time. Note that this agrees with the RFC 2822 format for time zone offsets, not the ISO 8601 format.

If both the :zone and :originalzone arguments are omitted, the local time zone MUST be used.

4.2. Date-part Argument

The date-part argument specifies a particular part of the resulting date/time value to match against the key-list. Possible case-insensitive values are:

```
  "year"      => the year, "0000" .. "9999".
  "month"     => the month, "01" .. "12".
  "day"       => the day, "01" .. "31".
  "date"      => the date in "yyyy-mm-dd" format.
  "julian"    => the Modified Julian Day, that is, the date
                 expressed as an integer number of days since
                 00:00 UTC on November 17, 1858 (using the Gregorian
                 calendar).  This corresponds to the regular
                 Julian Day minus 2400000.5.  Sample routines to
                 convert to and from modified Julian dates are
                 given in Appendix A.
  "hour"      => the hour, "00" .. "23".
  "minute"    => the minute, "00" .. "59".
  "second"    => the second, "00" .. "60".
  "time"      => the time in "hh:mm:ss" format.
  "iso8601"   => the date and time in restricted ISO 8601 format.
  "std11"     => the date and time in a format appropriate
                 for use in a Date: header field [RFC2822].
```

```
     "zone"      => the time zone in use.  If the user specified a
                    time zone with ":zone", "zone" will
                    contain that value.  If :originalzone is specified
                    this value will be the original zone specified
                    in the date-time value.  If neither argument is
                    specified the value will be the server's default
                    time zone in offset format "+hhmm" or "-hhmm".  An
                    offset of 0 (Zulu) always has a positive sign.
     "weekday"   => the day of the week expressed as an integer between
                    "0" and "6". "0" is Sunday, "1" is Monday, etc.
```

The restricted ISO 8601 format is specified by the date-time ABNF production given in [RFC3339], Section 5.6, with the added restrictions that the letters "T" and "Z" MUST be in upper case, and a time zone offset of zero MUST be represented by "Z" and not "+00:00".

4.3. Comparator Interactions with Date-part Arguments

Not all comparators are suitable with all date-part arguments. In general, the date-parts can be compared and tested for equality with either "i;ascii-casemap" (the default) or "i;octet", but there are two exceptions:

julian This is an integer, and may or may not have leading zeros. As such, "i;ascii-numeric" is almost certainly the best comparator to use with it.

std11 This is provided as a means to obtain date/time values in a format appropriate for inclusion in email header fields. The wide range of possible syntaxes for a std11 date/time -- which implementations of this extension are free to use when composing a std11 string -- makes this format a poor choice for comparisons. Nevertheless, if a comparison must be performed, this is case-insensitive, and therefore "i;ascii-casemap" needs to be used.

"year", "month", "day", "hour", "minute", "second" and "weekday" all use fixed-width string representations of integers, and can therefore be compared with "i;octet", "i;ascii-casemap", and "i;ascii-numeric" with equivalent results.

"date" and "time" also use fixed-width string representations of integers, and can therefore be compared with "i;octet" and "i;ascii-casemap"; however, "i;ascii-numeric" can't be used with it, as "i;ascii-numeric" doesn't allow for non-digit characters.

4.4. Examples

The Date: field can be checked to test when the sender claims to have created the message and act accordingly:

```
require ["date", "relational", "fileinto"];
if allof(header :is "from" "boss@example.com",
         date :value "ge" :originalzone "date" "hour" "09",
         date :value "lt" :originalzone "date" "hour" "17")
{ fileinto "urgent"; }
```

Testing the initial Received: field can provide an indication of when a message was actually received by the local system:

```
require ["date", "relational", "fileinto"];
if anyof(date :is "received" "weekday" "0",
         date :is "received" "weekday" "6")
{ fileinto "weekend"; }
```

5. Currentdate Test

```
Usage:   currentdate [":zone" <time-zone: string>]
                     [COMPARATOR] [MATCH-TYPE]
                     <date-part: string>
                     <key-list: string-list>
```

The currentdate test is similar to the date test, except that it operates on the current date/time rather than a value extracted from the message header. In particular, the ":zone" and date-part arguments are the same as those in the date test.

All currentdate tests in a single Sieve script MUST refer to the same point in time during execution of the script.

The :count value of a currentdate test is always 1.

5.1. Examples

The simplest use of currentdate is to have an action that only operates at certain times. For example, a user might want to have messages redirected to their pager after business hours and on weekends:

```
require ["date", "relational"];
if anyof(currentdate :is "weekday" "0",
         currentdate :is "weekday" "6",
         currentdate :value "lt" "hour" "09",
         currentdate :value "ge" "hour" "17")
{ redirect "pager@example.com"; }
```

Currentdate can be used to set up vacation [RFC5230] responses in advance and to stop response generation automatically:

```
require ["date", "relational", "vacation"];
if allof(currentdate :value "ge" "date" "2007-06-30",
         currentdate :value "le" "date" "2007-07-07")
{ vacation :days 7  "I’m away during the first week in July."; }
```

Currentdate may also be used in conjunction with the variables extension to pass time-dependent arguments to other tests and actions. The following Sieve places messages in a folder named according to the current month and year:

```
require ["date", "variables", "fileinto"];
if currentdate :matches "month" "*" { set "month" "${1}"; }
if currentdate :matches "year"  "*" { set "year"  "${1}"; }
fileinto "${month}-${year}";
```

Finally, currentdate can be used in conjunction with the editheader extension to insert a header-field containing date/time information:

```
 require ["variables", "date", "editheader"];
 if currentdate :matches "std11" "*"
   {addheader "Processing-date" "${0}";}
```

6. Index Extension

The "index" extension, if specified, adds optional :index and :last arguments to the header, address, and date tests as follows:

```
Syntax:   date [":index" <fieldno: number> [":last"]]
               [<":zone" <time-zone: string>> / ":originalzone"]
               [COMPARATOR] [MATCH-TYPE] <header-name: string>
               <date-part: string> <key-list: string-list>


Syntax:   header [":index" <fieldno: number> [":last"]]
                 [COMPARATOR] [MATCH-TYPE]
                 <header-names: string-list> <key-list: string-list>
```

```
Syntax:   address [":index" <fieldno: number> [":last"]]
                  [ADDRESS-PART] [COMPARATOR] [MATCH-TYPE]
                  <header-list: string-list> <key-list: string-list>
```

If :index <fieldno> is specified, the attempts to match a value are limited to the header field fieldno (beginning at 1, the first named header field). If :last is also specified, the count is backwards; 1 denotes the last named header field, 2 the second to last, and so on. Specifying :last without :index is an error.

:index only counts separate header fields, not multiple occurrences within a single field. In particular, :index cannot be used to test a specific address in an address list contained within a single header field.

Both header and address allow the specification of more than one header field name. If more than one header field name is specified, all the named header fields are counted in the order specified by the header-list.

6.1. Example

Mail delivery may involve multiple hops, resulting in the Received: field containing information about when a message first entered the local administrative domain being the second or subsequent field in the message. As long as the field offset is consistent, it can be tested:

```
# Implement the Internet-Draft cutoff date check assuming the
# second Received: field specifies when the message first
# entered the local email infrastructure.
require ["date", "relational", "index"];
if date :value "gt" :index 2 :zone "-0500" "received"
        "iso8601" "2007-02-26T09:00:00-05:00",
{ redirect "aftercutoff@example.org"; }
```

7. Security Considerations

The facilities defined here, like the facilities in the base Sieve specification, operate on message header information that can easily be forged. Note, however, that some fields are inherently more reliable than others. For example, the Date: field is typically inserted by the message sender and can be altered at any point. By contrast, the uppermost Received: field is typically inserted by the local mail system and is therefore difficult for the sender or an intermediary to falsify.

Use of the currentdate test makes script behavior inherently less predictable and harder to analyze. This may have consequences for systems that use script analysis to try and spot problematic scripts.

All of the security considerations given in the base Sieve specification also apply to these extensions.

8. IANA Considerations

The following templates specify the IANA registrations of the two Sieve extensions specified in this document:

```
To: iana@iana.org
Subject: Registration of new Sieve extensions

Capability name: date
Description:     The "date" extension gives Sieve the ability
                 to test date and time values.
RFC number:      RFC 5260
Contact address: Sieve discussion list <ietf-mta-filters@imc.org>

Capability name: index
Description:     The "index" extension provides a means to
                 limit header and address tests to specific
                 instances when more than one field of a
                 given type is present.
RFC number:      RFC 5260
Contact address: Sieve discussion list <ietf-mta-filters@imc.org>
```

9. References

9.1. Normative References

[CALGO199] Tantzen, R., "Algorithm 199: Conversions Between Calendar Date and Julian Day Number", Collected Algorithms from CACM 199.

[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.

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC5228] Guenther, P. and T. Showalter, "Sieve: An Email Filtering Language", RFC 5228, January 2008.

[RFC5231] Segmuller, W. and B. Leiba, "Sieve Email Filtering: Relational Extension", RFC 5231, January 2008.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

9.2. Informative References

[RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.

[RFC5230] Showalter, T. and N. Freed, "Sieve Email Filtering: Vacation Extension", RFC 5230, January 2008.

Appendix A. Julian Date Conversions

The following C routines show how to translate day/month/year information to and from modified Julian dates. These routines are straightforward translations of the Algol routines specified in CACM Algorithm 199 [CALGO199].

Given the day, month, and year, jday returns the modified Julian date.

```
int jday(int year, int month, int day)
{
    int j, c, ya;

    if (month > 2)
        month -= 3;
    else
    {
        month += 9;
        year--;
    }
    c = year / 100;
    ya = year - c * 100;
    return (c * 146097 / 4 + ya * 1461 / 4 + (month * 153 + 2) / 5 +
            day + 1721119);
}
```

Given j, the modified Julian date, jdate returns the day, month, and year.

```
void jdate(int j, int *year, int *month, int *day)
{
    int y, m, d;

    j -= 1721119;
    y = (j * 4 - 1) / 146097;
    j = j * 4 - y * 146097 - 1;
    d = j / 4;
    j = (d * 4 + 3) / 1461;
    d = d * 4 - j * 1461 + 3;
    d = (d + 4) / 4;
    m = (d * 5 - 3) / 153;
    d = d * 5 - m * 153 - 3;
    *day = (d + 5) / 5;
    *year = y * 100 + j;
    if (m < 10)
        *month = m + 3;
    else
    {
        *month = m - 9;
        *year += 1;
    }
}
```

Appendix B. Acknowledgements

Dave Cridland contributed the text describing the proper comparators to use with different date-parts. Cyrus Daboo, Frank Ellerman, Alexey Melnikov, Chris Newman, Dilyan Palauzov, and Aaron Stone provided helpful suggestions and corrections.

Author’s Address

Ned Freed
Sun Microsystems
800 Royal Oaks
Monrovia, CA 91016-6347
USA

Phone: +1 909 457 4293
EMail: ned.freed@mrochek.com