MARID
Internet-Draft
Expires: March 16, 2005

M. Wong
M. Lentczner

September 15, 2004

The SPF Record Format and Sender-ID Protocol
draft-ietf-marid-protocol-03

Status of this Memo

Copyright Notice

Abstract

This document defines a protocol for the authorization of Internet hosts to use domain names in the sender mailbox of mail that those hosts send. Authorization records are published in DNS for domain names that may be used as part of sender mailboxes. Mail receivers then perform a check against those records to see if a client host submitting a piece of mail is actually authorized. Since there are several different concepts of sender mailbox, this protocol is generic and can be applied to one or more of such "scopes".

Table of Contents

1. Introduction

Mail on the Internet suffers from a lack of authorization mechanisms. In particular, a host injecting mail into the mail stream can send mail using almost any mailbox as a sender in the envelope and headers. The Sender-ID protocol fills this void by enabling domains to authorize particular MTAs to send mail with their identity, and by enabling mail receivers to check these authorizations.

1.1 Publishing Domains

A Sender-ID compliant domain name is one with a valid, published SPF record. This record authorizes the use of the domain name, in one or more scopes (see below), by some sending MTAs, and not by others.

A compliant domain SHOULD publish authorizations for every defined scope.

Domain holders may publish SPF records that explicitly authorize no hosts for domain names that shouldn't be used in sender mailboxes.

1.2 Mail Receivers

A mail receiver can perform a Sender-ID compliant check, on one or more scopes, for each mail message it receives. Typically, these checks are done by a receiving MTA, but can be performed elsewhere in the mail processing chain so long as the required information is available.

It is expected that mail receivers will use the Sender-ID checks as part of the a larger set of tests on incoming mail. The results of other tests may influence whether or not a particular Sender-ID check is performed. For example, finding the sending host on a local white list may cause all other tests to be skipped and all mail from that host to be accepted.

When a mail receiver decides to perform a Sender-ID check, it MUST implement and evaluate the check_host() function (see below) correctly and follow the requirements of the particular scope under test. While the tests as a whole or optional, once it has been decided to perform a test it must as performed specified so that the correct semantics are preserved between publisher and receiver.

1.3 Scopes

There are several places in a mail transaction which involve the notion of a mail sender. In particular, the [RFC2821] envelope has a reverse-path, and the [RFC2822] headers have "From:", "Sender:",

"Resent-From:" and "Resent-Sender:". Since these identities have different semantics and processing characteristics (alone and in combination), Sender-ID defines different scopes. Publishing domains can make authorizations about one or all scopes, and mail receivers can check one or more scopes.

This document only defines the existence of two scopes: "mfrom" and "pra". The details of these two scopes are defined in other documents: "mfrom" is defined in [Mailfrom], "pra" is defined in [PRA].

Other scopes may be defined by future documents only. There is no registry for scopes. A scope definition must define what it identifies as the sending mailbox for a message, how to extract that information from a message, how to determine the initial arguments for the check_host() function, and what the compliant responses to the result are. This ensure that domains with published records and mail receiver agree on the semantics of the scope.

1.4 Earlier Work

This design is an evolution of work done under the name SPF. The record format and test presented here has been intentionally designed to be generally backward compatible with the currently deployed base of records and code.

1.5 Terminology

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].

2. SPF Records

SPF records to declare which hosts are, and are not, authorized to use a domain names for a given scope. Loosely, the record partitions all hosts into permitted and not-permitted sets. (Though some hosts might fall into other categories.)

The SPF record is a single string of text. An example record is:

```
spf2.0/mfrom,pra +mx +a:colo.example.com/28 -all
```

This record has a version of "spf2.0", a scope of "pra", and three directives: "+mx", "+a:colo.example.com/28", and "-all".

2.1 Publishing

A domain name's SPF record is published in DNS. The record is placed in the DNS tree at the domain name it pertains to.

The previous example might be published easily via this line in a domain zone file:

```
example.com. IN SPF "spf2.0/mfrom,pra +mx +a:colo.example.com/28
-all"
```

Note: The record is published at the domain name to which it pertains, not a name within the domain (such as is done with SRV records.) When published with via the SPF RR type (see below), this poses no problems and was chosen as the clearest way to express the declaration. When published via TXT records it is still published directly at the domain name, even though other TXT records, for other purposes may be published there.

2.1.1 RR Types

This document defines a new DNS RR type SPF, type code to be determined. The format of this type is identical to the TXT RR [RFC1035].

However, because there are a number of DNS server and resolver implementations in common use that cannot handle new RR types, a record can be published with type TXT.

A Sender-ID compliant domain name SHOULD have SPF records of both RR types. A domain name MUST have a record of at least one type. If a domain has records of both types, they MUST have identical content.

A Sender-ID compliant check SHOULD lookup both types. If both types

of records are returned for a domain, the SPF type MUST be used.

It is recognized that the current practice (using a TXT type record), is not optimal, but a practical reality due to the state of deployed software. The two record type scheme provides a forward path to the better solution of using a RR type reserved for this purpose.

For either type, the character content of the record is encoded as US-ASCII.

Example RRs in this document are shown with the SPF record type, however they could also be published with a TXT type.

2.1.2 Version

Each record starts with a version section. This version section contains a minor version field intended for future expansion. This document only defines records with a version section that starts "spf2.0" (minor version of "0").

2.1.3 Multiple Records

A domain name MUST NOT have multiple records that would cause an authorization check to select more than one record. See Section 3.5 for the selection rules.

In particular, a domain name cannot publish two or more records for any given scope where the version section differs only in the ver-minor field,

2.1.4 Additional Records

Some records contain directives that require additional SPF records. It is suggested that those records be placed under an "_spf" subdomain. See Appendix B for examples.

2.1.5 Multiple Strings

A Text DNS record (either TXT and SPF RR types) can be composed of more than one string. If a published record contains multiple strings, then record MUST be treated as if those strings are concatenated together without adding spaces. For example:

```
SPF "spf2.0/mfrom,pra ....  first" "second string..."
```

MUST be treated as equivalent to

```
SPF "spf2.0/mfrom,pra ....  firstsecond string..."
```

SPF or TXT records containing multiple strings are useful in order to construct longer records which would otherwise exceed the maximum length of a string within a TXT or SPF RR record.

Note: Some nameserver implementations will silently split long strings in TXT records into several shorter strings.

2.1.6 Record Size

All the published SPF records for a given domain name SHOULD remain small enough that the results of a query for them will fit within 512 octets. This will keep even older DNS implementations from falling over to TCP. Since the answer size is dependent on many things outside the scope of this document, it is only possible to give this guideline: If the combined length of the DNS name and the text of all the SPF records is under 480 characters, then DNS answers should fit in UDP packets. Note that when computing the sizes for queries of the TXT format, one must take into account any other TXT records published at the domain name.

2.1.7 Wildcard Records

Use of wildcard records for publishing is not recommended. Care must be taken if wildcard records are used. If a domain publishes wildcard MX records, it may want to publish wildcard declarations, subject to the same requirements and problems. In particular, the declaration must be repeated for any host that has any RR records at all, and for subdomains thereof. For example, the example given in [RFC1034], Section 4.3.3, could be extended with:

```
X.COM           MX      10      A.X.COM
X.COM           SPF     "spf2.0/mfrom,pra +a:A.X.COM -all"

*.X.COM         MX      10      A.X.COM
*.X.COM         SPF     "spf2.0/mfrom,pra +a:A.X.COM -all"

A.X.COM         A       1.2.3.4
A.X.COM         MX      10      A.X.COM
A.X.COM         SPF     "spf2.0/mfrom,pra +a:A.X.COM -all"

*.A.X.COM       MX      10      A.X.COM
*.A.X.COM       SPF     "spf2.0/mfrom,pra +a:A.X.COM -all"
```

Notice that the wildcard records must be repeated twice for every name within the domain: Once for the name, and once to cover the tree under the name.

Use of wildcards is discouraged in general as they cause every name under the domain to exist and queries against arbitrary names will never return RCODE 3 (Name Error).

2.1.8 Minor Version

This document only specifies records with a minor version of "0". All published records MUST start with "spf2.0".

Future versions of this document may define other minor versions to be used.

3. The check_host() Function

The check_host() function fetches SPF records, parses them, and interprets them to evaluate if a particular host is or is not permitted to send mail in a given context. Mail receivers that perform this check MUST correctly implement the check_host() function as described by the canonical algorithm defined here.

Implementations MAY use a different algorithm, so long as the results are the same.

3.1 Arguments

The function check_host() take four arguments:

```
<scope>  - the scope identifier
<ip>     - the IP address of the host under test
<domain> - the domain to check
<sender> - the full sending mailbox address
```

Each scope defines how these arguments are determined. See the individual scope documents [Mailfrom] and [PRA]

The domain portion of <sender> will usually be the same as the <domain> argument when check_host() is initially evaluated. However, it will generally not be true for recursive evaluations (see Section 4.2 below)

Note: The IP address may be either IPv4 or IPv6.

3.2 Results

The function check_host() can result in one of seven results described here. Based on the result, the action to be taken is determined by the checks a mail receiver is performing (see definitions for each scope) and the local policies of the receiver.

Results from interpreting valid records:

```
Neutral  (?): published data is explicitly inconclusive
Pass     (+): the <ip> is in the permitted set
Fail     (-): the <ip> is in the not permitted set
SoftFail (~): the <ip> may be in the not permitted set, its use is
              discouraged and the domain owner may move it to the not
              permitted set in the future
```

Results from error conditions:

```
None      - no published data
TempError - transient error during DNS lookup or other processing
PermError - unrecoverable error during processing, such as an
            error in the record format
```

If the result is "Fail", then an additional reason is returned. The reason may be one of:

```
Not Permitted
Malformed Domain
Domain Does Not Exist
```

If the reason is "Not Permitted", then an explanation string is also returned. The explanation string may be empty.

3.3 Initial Processing

If the <domain> is not an FQDN, the check_host() immediately returns the result "Fail" and a reason of "Malformed Domain".

If the <sender> has no localpart, substitute the string "postmaster" for the localpart.

3.4 Record Lookup

The records for <domain> are fetched. If the records are in a cache, and has not expired, then they may simply be used. Otherwise, the records must be fetched from DNS as follows:

In accordance with how the records are published, (see Section 2.1 above), a DNS query needs to be made for the <domain> name, querying for either RR type TXT, SPF or both.

If the domain does not exist (RCODE 3), check_host() exits immediately with the result "Fail" and a reason of "Domain Does Not Exist"

If the DNS lookup returns a server failure (RCODE 2), or other error (RCODE other than 0 or 3), or the query times out, check_host() exits immediately with the result "TempError"

3.5 Selecting Records

Records begin with version and scope sections:

```
record      = version scope terms *SP
version     = "spf2." ver-minor
ver-minor   = 1*DIGIT
scope       = "/" scope-id *( "," scope-id )
scope-id    = "mfrom" / "pra" / name
```

Starting with the set of records that were returned by the lookup, record selection proceeds in three steps:

1. If any records of type SPF are in the set, then all records of type TXT are discarded.
2. Records that do not begin with proper version and scope sections are discarded. The version section contains a ver-minor field that is for backward compatible future extensions. This field must be well-formed for a record to be retained, but is otherwise ignored.
3. Records that do not have a scope-id that matches <scope> are discarded. Note that this is a complete string match on the scope-id tokens: If <scope> is "pra", then the record starting "spf2.0/mfrom,prattle,fubar" would be discarded, but a record starting "spf2.0/mfrom,pra,fubar" would be retained.

After the above steps, there should be one record remaining and evaluation can proceed. If there are no records remaining, check_host() exits immediately with the result "None". If there are two or more records remaining, then check_host() exits immediately with the error "PermError".

3.6 Record Evaluation

After one SPF record has been selected, the check_host() function parses and interprets it to find a result for the current test. If at any point a syntax error is encountered, check_host() returns immediately with the result "PermError".

Implementations MAY choose to parse the entire record first and return "PermError" if the record is not well formed. See Section 6.1.

3.6.1 Term Evaluation

There are two types of terms: mechanisms and modifiers. A given mechanism type may always appear multiple times in a record. Modifiers may be constrained to appear at most once per record, depending on the definition of the modifier. Unknown mechanisms cause processing to abort with the result "PermError". Unknown modifiers are ignored.

A record contains an ordered list of mechanisms and modifiers:

```
terms       = *( 1*SP ( directive / modifier ) )

directive   = [ prefix ] mechanism
prefix      = "+" / "-" / "?" / "~"
mechanism   = name [ ":" domain-spec ] *( "/" *DIGIT )
modifier    = name "=" macro-string
name        = alpha *( alpha / digit / "-" / "_" / "." )
```

Most mechanisms allow a ":" or "/" character after the name.

Modifiers always contain an equals ('=') character immediately after the name, and before any ":" or "/" characters that may be part of the macro-string.

Terms that do not contain any of "=", ":" or "/" are mechanisms.

Mechanism and modifier names are case-insensitive. A mechanism "INCLUDE" is equivalent to "include".

3.6.2 Mechanisms

Each mechanism is considered in turn from left to right.

When a mechanism is evaluated, one of three things can happen: it can match, it can not match, or it can throw an exception.

If it matches, processing ends and the prefix value is returned as the result of that record. (The default prefix value is "+".)

If it does not match, processing continues with the next mechanism. If no mechanisms remain, the default result is specified in Section 3.7.

If it throws an exception, mechanism processing ends and the exception value is returned.

The possible prefixes, and the results they return are:
"+" Pass
"-" Fail
"~" SoftFail
"?" Neutral

A missing prefix for a mechanism is the same as a prefix of "+".

When a mechanism matches, and the prefix is "-" so that a "Fail" result is returned, the reason is Not Permitted, and the explanation

string is computed as described in Section 5.2.

Specific mechanisms are described in Section 4.

3.6.3 Modifiers

Modifiers are either global or positional:

Global modifiers MAY appear anywhere in the record, but SHOULD appear at the end, after all mechanisms and positional modifiers.

Positional modifiers apply only to the mechanism they follow. It is a syntax error for a positional modifier to appear before the first mechanism.

Modifiers of either type are also either singular or multiple:

Singular modifiers may appear only once in the record if they are global, or once after each mechanism if they are positional.

Multiple modifiers may appear multiple times in the record if they are global, or multiple times after each mechanism if they are positional.

The definition of each specific modifier (see Section 5) states whether it is global or positional, and whether it is singular or multiple. A modifier is not allowed to be defined as both global and positional.

Ordering of modifiers does not matter, except:
1) positional modifiers must appear after the mechanism they affect and before any subsequent mechanisms.
and 2) when a multiple modifier appears more than one time, the ordering of the appearances may be significant to the modifier.
Other than these constraints, implementations MUST treat different orders of modifiers the same. An intended side effect of these rules is modifiers cannot be defined that modify other modifiers.

These rules allow an implementation to correctly preparse a record. Furthermore, they are crafted to allow the parsing algorithm to be stable, even when new modifiers are introduced.

Modifiers which are unrecognized MUST be ignored. This allows older implementations to handle records with modifiers that were defined after they were written.

3.7 Default result

If none of the mechanisms match and there is no redirect modifier, then the check_host() exits with a result of "Neutral". If there is a redirect modifier, check_host() proceeds as defined in Section 5.1.

Note that records SHOULD always either use a redirect modifier or an "all" mechanism to explicitly terminate processing.

For example:

```
   spf2.0/mfrom,pra +mx -all
or
   spf2.0/mfrom,pra +mx redirect=_spf.example.com
```

3.8 Domain Spec

Several of these mechanisms and modifiers have a <domain-spec> section. The <domain-spec> string is macro expanded (see Section 7). The resulting string is the common presentation form of a fully qualified DNS name: A series of labels separated by periods. This domain is called the <target-name> in the rest of this document.

Note: The result of the macro expansion is not subject to any further escaping. Hence, this facility cannot produce all characters that are legal in a DNS label, for example, the space or control characters. However, this facility is powerful enough to express legal host names, and common utility labels (such as "_spf") that are used in DNS.

For mechanisms, the <domain-spec> is optional. If it is not provided, the <domain> is used as the <target-name>.

4. Mechanism Definitions

This section defines two types of mechanisms.

Basic mechanisms contribute to the language framework. They do not specify a particular type of authorization scheme.

- all
- include

Designated sender mechanisms are used to designate a set of <ip> address as being permitted or not to use the <domain> for sending mail.

- a
- mx
- ptr
- ip4
- ip6
- exists

Other mechanisms may be defined in the future.

Mechanisms either match, do not match, or throw an exception. If they match, their prefix value is returned. If they do not match, processing continues. If they throw an exception, the exception value is returned.

The following conventions apply to all mechanisms that perform an comparison between <ip> and an IP address at any point:

If no CIDR-length is given in the directive, then <ip> and the IP address are compared for equality.

If a CIDR-length is specified, then only the specified number of high-order bits of <ip> and the IP address are compared for equality.

When any mechanisms fetches host addresses to compare with <ip>, when <ip> is an IPv4 address, A records are fetched, when <ip> is an IPv6 address, AAAA records are fetched.

Several mechanisms rely on information fetched from DNS. While fetching that information and DNS server returns an error (RCODE other than 0 or 3) or the query times out, the mechanism throws the exception "TempError". Should the server return domain does not exist (RCODE 3), then evaluation of the mechanism continues with an empty set of records.

4.1 "all"

all = "all"

The "all" mechanism is a test that always matches. It is used as the rightmost mechanism in a record to provide an explicit default.

For example:

spf2.0/mfrom,pra +mx +a -all

Mechanisms after "all" will never be tested.

4.2 "include"

include = "include" ":" domain-spec

The "include" mechanism triggers a recursive evaluation of check_host(). The domain-spec is expanded as per Section 7. Then check_host() is evaluated with the resulting string as the <domain>. The <scope>, <ip> and <sender> arguments remain the same as current evaluation of check_host().

"Include" makes it possible for one domain to designate multiple administratively independent domains.

For example, a vanity domain "example.net" might send mail using the servers of administratively independent domains example.com and example.org.

Example.net could say

"spf2.0/mfrom,pra include:example.com include:example.org -all".

That would direct check_host() to, in effect, check the records of example.com and example.org for a "pass" result. Only if the host were not permitted for either of those domains would the result be "Fail".

Whether this mechanism matches or not, or throws an error depends on the result of the recursive evaluation of check_host():

A recursive check_host() result of:	Causes the include mechanism to:
Pass	match
Fail	not match
SoftFail	not match
Neutral	not match
TempError	throw TempError
PermError	throw PermError
None	throw PermError

The Include mechanism is intended for crossing administrative boundaries. While it is possible to use includes to consolidate multiple domains that share the same set of designated hosts, domains are encouraged to use redirects where possible, and to minimize the number of includes within a single administrative domain. For example, if example.com and example.org were managed by the same entity, and if the permitted set of hosts for both domains were "mx:example.com", it would be possible for example.org to specify "include:example.com", but it would be preferable to specify "redirect=example.com" or even "mx:example.com".

4.3 "a"

This mechanism matches if <ip> is one of the <target-name>'s IP addresses.

A = "a" [":" domain-spec] [dual-cidr-length]

An address lookup is done on the <target-name>. The <ip> is compared to the returned address(es). If any address matches, the mechanism matches.

4.4 "mx"

This mechanism matches if <ip> is one of the MX hosts for a domain name.

MX = "mx" [":" domain-spec] [dual-cidr-length]

check_host() first performs an MX lookup on the <target-name>. Then perform an address lookup on each MX name returned, in order of MX priority. The <ip> is compared to each returned IP address. If any address matches, the mechanism matches.

Note Regarding Implicit MXes: If the <target-name> has no MX records, check_host() MUST NOT pretend the target is its single MX, and MUST NOT default to an A lookup on the <target-name> directly. This behavior breaks with the legacy "implicit MX" rule. See [RFC2821] Section 5. If such behavior is desired, the publisher should specify an "a" directive.

4.5 "ptr"

This mechanism tests if the DNS reverse mapping for <ip> exists and validly points to a domain name within a particular domain.

PTR = "ptr" [":" domain-spec]

First the <ip>'s name is looked up using this procedure: perform a DNS reverse-mapping for <ip>, looking up the corresponding PTR record in "in-addr.apra.". For each record returned, validate the host name by looking up its IP address. If <ip> is among the returned IP addresses, then that host name is validated. In pseudocode:

```
sending-host_names := ptr_lookup(sending-host_IP);
for each name in (sending-host_names) {
  IP_addresses := a_lookup(name);
  if the sending-host_IP is one of the IP_addresses {
    validated_sending-host_names += name;
  }
}
```

Check all validated hostnames to see if they end in the <target-name> domain. If any do, this mechanism matches. If no validated hostname can be found, or if none of the validated hostnames end in the <target-name>, this mechanism fails to match.

Pseudocode:

```
for each name in (validated_sending-host_names) {
  if name ends in <domain-spec>, return match.
  if name is <domain-spec>, return match.
}
return no-match.
```

This mechanism matches if the <target-name> is an ancestor of a validated hostname, or if the <target-name> and a validated hostname;

are the same. For example: "mail.example.com" is within the domain "example.com", but "mail.bad-example.com" is not. If a validated hostname is the <target-name>, a match results.

Note: This mechanism is not recommended. If a domain decides to use it, it should make sure is has the proper PTR records in place for its hosts.

4.6 "ip4" and "ip6"

These mechanisms test if <ip> is contained within a given IP network.

```
IP4             = "ip4" ":" ip4-network [ ip4-cidr-length ]
IP6             = "ip6" ":" ip6-network [ ip6-cidr-length ]
ip4-cidr-length = "/" 1*DIGIT
ip6-cidr-length = "/" 1*DIGIT

ip4-network     = as per conventional dotted quad notation,
                  e.g. 192.0.2.0
ip6-network     = as per [RFC 3513], section 2.2,
                  e.g. 2001:DB8::CD30
```

The <ip> is compared to the given network. If CIDR-length high-order bits match, the mechanism matches.

If ip4-cidr-length is omitted it is taken to be "/32". If ip6-cidr-length is omitted it is taken to be "/128".

4.7 "exists"

This mechanism is used to construct an arbitrary host name that is used for a DNS A record query. It allows for complicated schemes involving arbitrary parts of the mail envelope to determine what is legal.

```
exists = "exists" ":" domain-spec
```

The domain-spec is expanded as per Section 7. The resulting domain name is used for a DNS A lookup. If any A record is returned, this mechanism matches. The lookup type is 'A' even when the connection type is IPv6.

Domains can use this mechanism to specify arbitrarily complex queries. For example, suppose example.com publishes the record:

```
spf2.0/mfrom,pra exists:%{ir}.%{l1r+-}._spf.%{d} -all
```

The target-name might expand to

"1.2.0.192.someuser._spf.example.com". This makes fine-grained decisions possible at the level of the user and client IP address.

This mechanism enable queries that mimic the style of tests that existing RBL lists use.

5. Modifier Definitions

Modifiers are not mechanisms: they do not return match or no-match. Instead they provide additional information or alter check_host() processing. Global modifiers affect the entire record, whereas positional modifiers only affect the preceding mechanism in the record.

While unrecognized mechanisms cause an immediate "PermError" abort, unrecognized modifiers MUST be simply ignored. Modifiers therefore provide an way to extend the record format in the future with backward compatibility.

Only two modifiers are currently defined: "redirect" and "exp". Implementations of check_host() MUST support them both.

This document reserves two modifiers for future definition: "accredit" and "match_subdomains". Until these are defined, implementations SHOULD ignore them. There is also one deprecated modifier: "default". Implementations MUST ignore it.

5.1 redirect: Redirected Query

The "redirect" modifier is global and singular.

If all mechanisms fail to match, and a redirect modifier is present, then processing proceeds as follows.

redirect = "redirect" "=" domain-spec

The domain-spec portion of the redirect section is expanded as per the macro rules in Section 7. Then check_host() is evaluated with the resulting string as the <domain>. The <scope>, <ip> and <sender> arguments remain the same as current evaluation of check_host().

The result of this new evaluation of check_host() is then considered the result of current evaluation.

Note that the newly queried domain may itself specify redirect processing.

This facility is intended for use by organizations that wish to apply the same record to multiple domains. For example:

```
  la.example.com. SPF "spf2.0/mfrom,pra redirect=_spf.example.com"
  ny.example.com. SPF "spf2.0/mfrom,pra redirect=_spf.example.com"
  sf.example.com. SPF "spf2.0/mfrom,pra redirect=_spf.example.com"
_spf.example.com. SPF "spf2.0/mfrom,pra mx:example.com -all"
```

In this example, mail from any of the three domains is described by the same record. This can be an administrative advantage.

Note: In general, a domain A cannot reliably use a redirect to another domain B not under the same administrative control. Since the <sender> stays the same, there is no guarantee that the record at domain B will correctly work for addresses in domain A, especially if domain B uses mechanisms involving localparts. An "include" directive may be more appropriate.

For clarity it is RECOMMENDED that any redirect modifier appear as the very last term in a record.

5.2 exp: Explanation

The "exp" modifier is global and singular.

explanation = "exp" "=" domain-spec

If check_host() results in a "Fail" due to a mechanism match (such as "-all"), and the exp modifier is present, then the explanation string returned is computed as described below. If no exp modifier is present, then an empty explanation string is returned.

The <domain-spec> is macro expanded (see Section 7) and becomes the <target-name>. The DNS TXT record for the <target-name> is fetched either from a cache or via a query to DNS.

If <domain-spec> is empty, or there are any processing errors (any RCODE other than 0), or if no records are returned, or if more than one record is returned, then an empty explanation string is returned.

The fetched TXT record's strings are concatenated with no spaces, and then treated as a new macro-string which is macro-expanded. This final result is the explanation string.

Software evaluating check_host() can use this string when the result is "Fail" with a reason of "Not Permitted", to communicate information from the publishing domain in the form of a short message or URL. Software should make it clear that the explanation string comes from a third party. For example, it can prepend the macro string "%{d} explains: " to the explanation.

Implementations MAY limit the length of the resulting explanation string to allow for other protocol constraints and/or reasonable processing limits.

Suppose example.com has this record

spf2.0/mfrom,pra mx -all exp=explain._spf.%{d}

Here are some examples of possible explanation TXT records at explain._spf.example.com:

Example.com mail should only be sent by its own servers.

-- a simple, constant message

%{i} is not one of %{d}'s designated mail servers.

-- a message with a little more info, including the IP address that failed the check

See http://%{d}/why.html?s=%{S}&i=%{I}

-- a complicated example that constructs a URL with the parameters check_host() so that a web page can be generated with detailed, custom instructions

Note: During recursion into an Include mechanism, explanations do not propagate out. But during execution of a Redirect modifier, the explanation string from the target of the redirect is used.

6. Miscellaneous

6.1 Unrecognized Mechanisms and Modifiers

New mechanisms can only be introduced by new versions of this document.

Unrecognized mechanisms cause processing to abort: If, during evaluation of a record, check_host() encounters a mechanism which it does not understand, it terminates processing and returns "PermError", without evaluating any further mechanisms. Mechanisms listed before the unknown mechanism MUST, however, be evaluated.

For example, consider the record:

```
spf2.0/mfrom,pra a mx ptr foo:_foo.%{d} -all
```

If during the evaluation of check_host(), any of the "a", "mx", or "ptr" directives match, then check_host() would return a "Pass" result. If none of those directives resulted in a match, then an implementation that did not recognize the "foo" mechanism would return "PermError". An implementation that did recognize the "foo" mechanism would be able to perform an extended evaluation.

Note: "foo" is an example of an unknown extension mechanism that could be defined in the future. It is NOT defined by this proposal.

New modifiers can be introduced by registering them with the IANA, or in new versions of this document

Unrecognized modifiers are ignored: if an implementation encounters modifiers which it does not recognize, it MUST ignore them.

6.2 Processing Limits

During processing, an evaluation of check_host() may require additional evaluations of check_host() due to the Include mechanism and/or the Redirect modifier.

Implementations must be prepared to handle records that are set up incorrectly or maliciously. Implementations MUST perform loop detection, limit additional evaluations, or both. If an implementation chooses to limit additional evaluations, then at least a total of 10 evaluations of check_host() for a single query MUST be supported. (This number should be enough for even the most complicated configurations.)

If a loop is detected, or evaluation limit of an implementation is

reached, check_host() MUST abort processing and return the result "PermError".

MTAs or other processors MAY also impose a limit on the maximum amount of elapsed time to evaluate check_host(). Such a limit SHOULD allow at least 20 seconds. If such a limit is exceeded, the result of authentication SHOULD be "TempError".

Domains publishing records SHOULD try keep the number include directives, and chained redirect modifiers to a minimum. Domains SHOULD also try to minimize the amount of other DNS information needed to evaluate a record. This can be done by choosing directives that require less DNS information.

For example, consider a domain set up as:

```
example.com.      IN MX   10 mx.example.com.
mx.example.com.   IN A    192.0.2.1
a.example.com.    IN SPF  "spf2.0/mfrom,pra +mx:example.com -all"
b.example.com.    IN SPF  "spf2.0/mfrom,pra +a:mx.example.com -all"
c.example.com.    IN SPF  "spf2.0/mfrom,pra +ip4:192.0.2.1 -all"
```

Evaluating check_host() for the domain "a.example.com" requires the MX records for "example.com", and then the A for records for the listed hosts. Evaluating for "b.example.com" only requires the A records. Evaluating for "c.example.com" requires none.

However, there may be administrative considerations: Using "a" over "ip4" allows hosts to be renumbered easily. Using "mx" over "a" allows the set of mail hosts to be changed easily.

7. Macros

7.1 Macro definitions

Certain terms perform macro interpolation on their arguments.

```
domain-spec  = *( macro-char / v-char-dm )
macro-string = *( macro-char / v-char-ms )
macro-char   = ( "%{" ALPHA transformer *delimiter "}" )
               / "%%" / "%_" / "%-"
transformer  = *DIGIT [ "r" ]
delimiter    = "." / "-" / "+" / "," / "/" / "_" / "="
v-char-dm    = %x21-24 / %x26-2E / %x30-7E
               ; visible characters except "%" and "/"
v-char-ms    = %x21-24 / %x26-7E
               ; visible characters except "%"
```

A literal "%" is expressed by "%%".
"%_" expands to a single " " space.
"%-" expands to a URL-encoded space, viz. "%20".

The following macro letters are expanded in term arguments:

- s = <sender>
- l = local-part of <sender>
- o = domain of <sender>
- d = <domain>
- i = <ip>
- p = the validated host name of <ip>
- v = the string "in-addr" for if <ip> is ipv4, or "ip6" if <ip> is ipv6

The following macro letters are only allowed in "exp" text:

- c = SMTP client IP (easily readable format)
- r = domain name of host performing the check
- t = current timestamp in UTC epoch seconds notation

The uppercase versions of all these macros are URL-encoded.

A '%' character not followed by a '{', '%', '-', or '_' character MUST be interpreted as a literal. Domains SHOULD NOT rely on this feature; they MUST escape % literals. For example, an explanation TXT record

> Your spam volume has increased by 581%

is incorrect. Instead, say

Your spam volume has increased by 581%%

All other legal visible characters are simply expanded to themselves. Note that the two different macro contexts, domain-spec, and macro-string allow slightly different sets of legal visible characters, v-char-dm and v-char-ms respectively.

Legal optional transformers are:

*DIGIT : zero or more digits
'r' : reverse value, splitting on dots by default

If transformers or delimiters are provided, the replacement value for a macro letter is split into parts. After performing any reversal operation and/or removal of left-hand parts, the parts are rejoined using "." and not the original splitting characters.

By default, strings are split on "." (dots). Note that no special treatment is given to leading, trailing or consecutive delimiters, and so the list of parts may contain empty strings. Macros may specify delimiter characters which are used instead of ".". Delimiters MUST be one or more of the characters:

"." / "-" / "+" / "," / "/" / "_" / "="

The 'r' transformer indicates a reversal operation: if the client IP address were 192.0.2.1, the macro %{i} would expand to "192.0.2.1" and the macro %{ir} would expand to "1.2.0.192".

The DIGIT transformer indicates the number of right-hand parts to use, after optional reversal. If a DIGIT is specified, the value MUST be nonzero. If no DIGITs are specified, or if the value specifies more parts than are available, all the available parts are used. If the DIGIT was 5, and only 3 parts were available, the macro interpreter would pretend the DIGIT was 3. Implementations MUST support at least a value of 128, as that is the maximum number of labels in a domain name.

The "s" macro expands to the <sender> argument. It is an email address with a localpart, an "@" character, and a domain. The "l" macro expands to just the localpart. The "o" macro expands to just the domain part. Note that these values remain the same during a recursive and chained evaluations due to "include" and/or "redirect". Note also that if the original <sender> had no localpart, the localpart was set to "postmaster" in initial processing (see Section 3.3).

For IPv4 addresses, both the "i" and "c" macros expand to the standard dotted-quad format.

For IPv6 addresses, the "i" macro expands to dot-format address; it is intended for use in %{ir}. The "c" macro may expand to any of the hexadecimal colon-format addresses specified in [RFC3513] section 2.2. It is intended for humans to read.

The "p" macro expands to the validated host name of <ip>. The procedure for finding the validated host names is defined in Section 4.5. If that procedure produces more than one validated host name, any name from the list may be used. If that procedure produces no validate host name the string "unknown" is used.

The "r" macro expands to the name of the receiving MTA. This SHOULD be a fully qualified domain name, but if one does not exist (as when the checking is done by a script) or if policy restrictions dictate otherwise, the word "unknown" SHOULD be substituted. The domain name MAY be different than the name found in the MX record that the client MTA used to locate the receiving MTA.

Any unrecognized macro letters are expanded as the string "unknown". There is one deprecated macro letter: "h". It is expanded as the string "deprecated".

When the result of macro expansion is used in a domain name query, if the expanded domain name exceeds 255 characters (the maximum length of a domain name), the left side is truncated to fit, by removing successive subdomains until the total length falls below 255 characters.

Uppercased macros expand exactly as their lower case equivalents, and are then URL escaped. URL escaping is described in [RFC2396].

Note: Domains should avoid using the "s", "l" or "o" macros in conjunction with any mechanism directive. While these macros are powerful and allow per-user records to be published, they severely limit the ability of implementations to cache results of check_host() and they reduce the effectiveness of DNS caches.

Implementations should be aware that if no directive processed during the evaluation of check_host() contains an "s", "l", or "o" macro, then the results of the evaluation can be cached on the basis of <domain> and <ip> alone for as long as the shortest TTL of all the DNS records involved.

7.2 Expansion Examples

```
   The <sender> is strong-bad@email.example.com.
   The IPv4 SMTP client IP is 192.0.2.3.
   The IPv6 SMTP client IP is 5f05:2000:80ad:5800::1.
   The PTR domain name of the client IP is mx.example.org.

macro                       expansion
-------  ---------------------------
%{s}     strong-bad@email.example.com
%{o}                email.example.com
%{d}                email.example.com
%{d4}               email.example.com
%{d3}               email.example.com
%{d2}                    example.com
%{d1}                            com
%{dr}               com.example.email
%{d2r}                  example.email
%{l}                       strong-bad
%{l-}                      strong.bad
%{lr}                      strong-bad
%{lr-}                     bad.strong
%{l1r-}                        strong


macro-string                                              expansion
-------------------------------------------------------------------
%{ir}.%{v}._spf.%{d2}             3.2.0.192.in-addr._spf.example.com
%{lr-}.lp._spf.%{d2}                  bad.strong.lp._spf.example.com

%{lr-}.lp.%{ir}.%{v}._spf.%{d2}
                     bad.strong.lp.3.2.0.192.in-addr._spf.example.com

%{ir}.%{v}.%{l1r-}.lp._spf.%{d2}
                         3.2.0.192.in-addr.strong.lp._spf.example.com

%{d2}.trusted-domains.example.net
                              example.com.trusted-domains.example.net

IPv6:
%{ir}.%{v}._spf.%{d2}          1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.8.
                       5.d.a.0.8.0.0.0.2.5.0.f.5.ip6._spf.example.com
```

8. Security Considerations

There are two aspects of this protocol that malicious parties could exploit to undermine the validity of the check_host() function:

The evaluation of check_host() relies heavily on DNS. A malicious attacker could poison a target's DNS cache with spoofed DNS data, and cause check_host() to return incorrect results, including "Pass" for an <ip> value where the actual domain's record would evaluate to "Fail".

The client IP address, <ip>, is assumed to be correct true. A malicious attacker could spoof TCP sequences to make mail appear to come from a permitted host for a domain that the attacker is impersonating.

As with most aspects of mail, there are a number of ways that malicious parties could use the protocol as an avenue of a distributed denial of service attack:

While implementations of check_host() need to limit the number of includes and redirects and/or check for loops, malicious domains could publish records that exercise or exceed these limits in an attempt to waste computation effort at their targets when they send them mail.

Malicious parties could send large volume mail purporting to come from the intended target to a wide variety of legitimate mail hosts. These legitimate machines would then present a DNS load on the target as they fetched the relevant records.

While these distributed denial of service attacks are possible, they seem more convoluted to mount, and have less of an impact, than other simpler attacks.

The exp modifier allows the domain being checked to provide a text message should the check_host() function fail. Explicit provisions in the macro facility support domains including URLs in this message. Since this message is eventually shown to a user, that user needs to take care, as with all URLs, in deciding to follow the URL.

9. IANA Considerations

The IANA needs to assign a new Resource Record Type and Qtype from the DNS Parameters Registry for the SPF RR type.

The IANA will need to maintain one registry in support of this specification. The registry consists of the modifiers as described in Section 3.6.3 and Section 5

[[Missing application review policy]]

9.1 Registration Template

To: ietf-types@iana.org
Subject: Registration of SPF Modifier XXX

Modifier name:

Type: (Global or Positional)

Appearance: (Single or Multiple)

Security considerations:

Interoperability considerations:

Published specification:

Person & email address to contact for further information:

Author/Change controller:

(Any other information that the author deems interesting may be added below this line.)

[[This template needs to be reviewed]]

10. Contributors and Acknowledgements

This design owes a debt of parentage to RMX (by Hadmut Danisch in 2003) and to DMP (by Gordon Fecyk in 2003). It traces its ancestry farther back through "Repudiating Mail-From" by Paul Vixie in 2002 to a suggestion by Jim Miller in 1998.

Philip Gladstone contributed macros to the specification, multiplying the expressiveness of the language and making per-user and per-IP lookups possible.

The authors would also like to thank the literally hundreds of individuals who have participated in the development of this design. There are far too numerous to name, but they include:

The folks on the SPAM-L mailing list.
The folks on the ASRG and MARID/MXCOMP mailing lists.
The folks on the spf-discuss mailing list.
The folks on the mailing list that shall not be named.
The folks on #perl.

11. Comments

Comments on this draft are welcome. In the interests of openness, rather than contacting the authors directly, please post to either:

the ietf-mxcomp mailing list
http://www.imc.org/ietf-mxcomp/index.html

or

the spf-discuss mailing list.
http://spf.pobox.com/mailinglist.html

12. References

12.1 Normative References

[RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.

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

[RFC2234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[RFC2396] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[RFC3513] Hinden, R. and S. Deering, "Internet Protocol Version 6 (IPv6) Addressing Architecture", RFC 3513, April 2003.

12.2 Informative References

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.

[RFC2821] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

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

[RFC3668] Bradner, S., "Intellectual Property Rights in IETF Technology", BCP 79, RFC 3668, February 2004.

[Mailfrom] Lentczner, M. and M. Wong, "Authorizing Use of Domains in MAIL FROM", draft-ietf-marid-mailfrom-00 (work in progress), September 2004.

[PRA] Lyon, J. and M. Wong, "Sender ID: Authenticating E-Mail", draft-ietf-marid-core-03 (work in progress), August 2004.

[RMX] Danish, H., "The RMX DNS RR Type for light weight sender authentication", October 2003.

Work In Progress

[DMP] Fecyk, G., "Designated Mailers Protocol", December 2003.

Work In Progress

[Vixie] Vixie, P., "Repudiating Mail-From", 2002.

Authors' Addresses

Meng Weng Wong
Singapore

EMail: mengwong+spf@pobox.com

Mark Lentczner
1209 Villa Street
Mountain View, CA 94041
United States of America

EMail: markl@glyphic.com
URI: http://www.ozonehouse.com/mark/

Appendix A. Collected ABNF

This section is normative and any discrepancies with the ABNF fragments in the preceding text are to be resolved in favor of this grammar.

See [RFC2234] for ABNF notation.

```
record      = version scope terms *SP

version     = "spf2." ver-minor
ver-minor   = 1*DIGIT
scope       = "/" scope-id *[ "," scope-id ]
scope-id    = "mfrom" / "pra" / name

terms       = *( 1*SP ( directive / modifier ) )

directive   = [ prefix ] mechanism
prefix      = "+" / "-" / "?" / "~"
mechanism   = ( all / include
               / A / MX / PTR / IP4 / IP6 / exists
               / unknown-mechanism )

all          = "all"
include      = "include"  ":" domain-spec
A            = "a"      [ ":" domain-spec ] [ dual-cidr-length ]
MX           = "mx"     [ ":" domain-spec ] [ dual-cidr-length ]
PTR          = "ptr"    [ ":" domain-spec ]
IP4          = "ip4"      ":" ip4-network   [ ip4-cidr-length ]
IP6          = "ip6"      ":" ip6-network   [ ip6-cidr-length ]
exists       = "exists"   ":" domain-spec

unknown-mechanism = name [ ":" macro-string ]

modifier    = redirect / explanation / unknown-modifier
redirect    = "redirect" "=" domain-spec
explanation = "exp" "=" domain-spec
unknown-modifier = name "=" macro-string


ip4-network  = as per conventional dotted quad notation,
               e.g. 192.0.2.0
ip6-network  = as per [RFC 3513], section 2.2,
               e.g. 2001:DB8::CD30

dual-cidr-length = [ ip4-cidr-length ] [ "/" ip6-cidr-length ]
ip4-cidr-length  = "/" 1*DIGIT
ip6-cidr-length  = "/" 1*DIGIT
```

```
   domain-spec  = *( macro-char / v-char-dm )
   macro-string = *( macro-char / v-char-ms )
   macro-char   = ( "%{" ALPHA transformer *delimiter "}" )
                  / "%%" / "%_" / "%-"
   transformer  = *DIGIT [ "r" ]

   name        = ALPHA *( ALPHA / DIGIT / "-" / "_" / "." )
   delimiter   = "." / "-" / "+" / "," / "/" / "_" / "="
   v-char-dm   = %x21-24 / %x26-2E / %x30-7E
                 ; visible characters except "%" and "/"
   v-char-ms   = %x21-24 / %x26-7E
                 ; visible characters except "%"
```

Appendix B. Extended Examples

These examples are based on the following DNS setup:

```
; A domain with two mail servers, two hosts
; and two servers at the domain name

$ORIGIN example.com.
@           MX  10 mail-a
            MX  20 mail-b
            A   192.0.2.10
            A   192.0.2.11
amy         A   192.0.2.65
bob         A   192.0.2.66
mail-a      A   192.0.2.129
mail-b      A   192.0.2.130
www         CNAME example.com.

; A related domain

$ORIGIN example.org
@           MX  10 mail-c
mail-c      A   192.0.2.140

; The reverse IP for those addresses

$ORIGIN 2.0.192.in-addr.arpa.
10          PTR example.com.
11          PTR example.com.
65          PTR amy.example.com.
66          PTR bob.example.com.
129         PTR mail-a.example.com.
130         PTR mail-b.example.com.
140         PTR mail-c.example.org.

; A rogue reverse IP domain that claims to be
; something it's not

$ORIGIN 0.0.10.in-addr.arpa.
4           PTR bob.example.com.
```

B.1 Simple Examples

These examples show various possible published records for example.com and which values if <ip> would cause check_host() to return "Pass". Note that <domain> is "example.com".

```
spf2.0/mfrom,pra +all
   -- any <ip> passes

spf2.0/mfrom,pra a -all
   -- hosts 192.0.2.10 and 192.0.2.11 pass

spf2.0/mfrom,pra a:example.org -all
   -- no sending hosts pass since example.org has no A records

spf2.0/mfrom,pra mx -all
   -- sending hosts 192.0.2.129 and 192.0.2.130 pass

spf2.0/mfrom,pra mx:example.org -all
   -- sending host 192.0.2.140 passes

spf2.0/mfrom,pra mx mx:example.org -all
   -- sending hosts 192.0.2.129, 192.0.2.130, and 192.0.2.140 pass

spf2.0/mfrom,pra mx/30 mx:example.org/30 -all
   -- any sending host in 192.0.2.128/30 or 192.168.2.140/30 passes

spf2.0/mfrom,pra ptr -all
   -- sending host 192.0.2.65 passes (reverse IP is valid and in
   example.com)
   -- sending host 192.0.2.140 fails (reverse IP is valid, but not in
   example.com)
   -- sending host 10.0.0.4 fails (reverse IP is not valid)

spf2.0/mfrom,pra ip4:192.0.2.128/28 -all
   -- sending host 192.0.2.65 fails
   -- sending host 192.0.2.129 passes
```

B.2 Multiple Domain Example

These examples show the effect of related records:

```
   example.org: "spf2.0/mfrom,pra include:example.com
   include:example.net -all"
```

This record would be used if mail from example.org actually came through servers at example.com and example.net. Example.org's designated servers are the union of example.com and example.net's designated servers.

```
   la.example.org: "spf2.0/mfrom,pra redirect=example.org"
   ny.example.org: "spf2.0/mfrom,pra redirect=example.org"
   sf.example.org: "spf2.0/mfrom,pra redirect=example.org"
```

These records allow a set of domains that all use the same mail system to make use of that mail system's record. In this way, only the mail system's record needs to updated when the mail setup changes. These domains' records never have to change.

B.3 RBL Style Example

Imagine that, in addition to the domain records listed above, there are these:

```
$Origin _spf.example.com.
mary.mobile-users                   A 127.0.0.2
fred.mobile-users                   A 127.0.0.2
15.15.168.192.joel.remote-users     A 127.0.0.2
16.15.168.192.joel.remote-users     A 127.0.0.2
```

The following records describe users at example.com who mail from arbitrary servers, or who mail from personal servers.

example.com:

```
spf2.0/mfrom,pra mx
       include:mobile-users._spf.%{d}
       include:remote-users._spf.%{d}
       -all
```

mobile-users._spf.example.com:

```
spf2.0/mfrom,pra exists:%{l1r+}.%{d}
```

remote-users._spf.example.com:

```
spf2.0/mfrom,pra exists:%{ir}.%{l1r+}.%{d}
```

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.