Using TLS in Applications   D. Margolis
Internet-Draft   M. Risher
Intended status: Standards Track   Google, Inc
Expires: October 6, 2018   B. Ramakrishnan
                                                         Yahoo!, Inc
                                                          A. Brotman
                                                        Comcast, Inc
                                                            J. Jones
                                                      Microsoft, Inc
                                                       April 4, 2018

              SMTP MTA Strict Transport Security (MTA-STS)
                        draft-ietf-uta-mta-sts-15

Abstract

   SMTP Mail Transfer Agent Strict Transport Security (MTA-STS) is a
   mechanism enabling mail service providers to declare their ability to
   receive Transport Layer Security (TLS) secure SMTP connections, and
   to specify whether sending SMTP servers should refuse to deliver to
   MX hosts that do not offer TLS with a trusted server certificate.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at http://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on October 6, 2018.

Copyright Notice

publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.

Table of Contents

1. Introduction

   The STARTTLS extension to SMTP [RFC3207] allows SMTP clients and
   hosts to negotiate the use of a TLS channel for encrypted mail
   transmission.

   While this opportunistic encryption protocol by itself provides a
   high barrier against passive man-in-the-middle traffic interception,
   any attacker who can delete parts of the SMTP session (such as the
   "250 STARTTLS" response) or who can redirect the entire SMTP session
   (perhaps by overwriting the resolved MX record of the delivery
   domain) can perform downgrade or interception attacks.

   This document defines a mechanism for recipient domains to publish
   policies specifying:

   o   whether MTAs sending mail to this domain can expect PKIX-
       authenticated TLS support

   o   what a conforming client should do with messages when TLS cannot
       be successfully negotiated

1.1. 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]. These
   words may also appear in this document in lowercase, absent their
   normative meanings.

   We also define the following terms for further use in this document:

   o   MTA-STS Policy: A commitment by the Policy Domain to support PKIX
       [RFC5280] authenticated TLS for the specified MX hosts.

   o   Policy Domain: The domain for which an MTA-STS Policy is defined.
       This is the next-hop domain; when sending mail to
       "alice@example.com" this would ordinarily be "example.com", but
       this may be overridden by explicit routing rules (as described in
       Section 3.4, "Policy Selection for Smart Hosts and Subdomains").

2. Related Technologies

   The DANE TLSA record [RFC7672] is similar, in that DANE is also
   designed to upgrade unauthenticated encryption or plaintext
   transmission into authenticated, downgrade-resistant encrypted
   transmission. DANE requires DNSSEC [RFC4033] for authentication; the
   mechanism described here instead relies on certificate authorities

(CAs) and does not require DNSSEC, at a cost of risking malicious
downgrades. For a thorough discussion of this trade-off, see
Section 10, "Security Considerations".

In addition, MTA-STS provides an optional testing-only mode, enabling
soft deployments to detect policy failures; partial deployments can
be achieved in DANE by deploying TLSA records only for some of a
domain's MXs, but such a mechanism is not possible for the per-domain
policies used by MTA-STS.

The primary motivation of MTA-STS is to provide a mechanism for
domains to ensure transport security even when deploying DNSSEC is
undesirable or impractical. However, MTA-STS is designed not to
interfere with DANE deployments when the two overlap; in particular,
senders who implement MTA-STS validation MUST NOT allow a "valid" or
"testing"-only MTA-STS validation to override a failing DANE
validation.

3. Policy Discovery

MTA-STS policies are distributed via HTTPS from a "well-known"
[RFC5785] path served within the Policy Domain, and their presence
and current version are indicated by a TXT record at the Policy
Domain. These TXT records additionally contain a policy "id" field,
allowing sending MTAs to check the currency of a cached policy
without performing an HTTPS request.

To discover if a recipient domain implements MTA-STS, a sender need
only resolve a single TXT record. To see if an updated policy is
available for a domain for which the sender has a previously cached
policy, the sender need only check the TXT record's version "id"
against the cached value.

3.1. MTA-STS TXT Records

The MTA-STS TXT record is a TXT record with the name "_mta-sts" at
the Policy Domain. For the domain "example.com", this record would
be "_mta-sts.example.com". MTA-STS TXT records MUST be US-ASCII,
semicolon-separated key/value pairs containing the following fields:

o "v": (plain-text, required). Currently only "STSv1" is supported.

o "id": (plain-text, required). A short string used to track policy
   updates. This string MUST uniquely identify a given instance of a
   policy, such that senders can determine when the policy has been
   updated by comparing to the "id" of a previously seen policy.
   There is no implied ordering of "id" fields between revisions.

An example TXT record is as below:

"_mta-sts.example.com. IN TXT "v=STSv1; id=20160831085700Z;"""

The formal definition of the "_mta-sts" TXT record, defined using
[RFC7405], is as follows:

sts-text-record = sts-version 1*(field-delim sts-field) [field-delim]

sts-field   = sts-id /   ; Note that sts-id record
                  sts-extension   ; is required.

field-delim   = *WSP ";" *WSP

sts-version   = %s"v=STSv1"

sts-id   = %s"id=" 1*32(ALPHA / DIGIT)   ; id=...

sts-extension   = sts-ext-name "=" sts-ext-value ; name=value

sts-ext-name   = (ALPHA / DIGIT)
                  *31(ALPHA / DIGIT / "_" / "-" / ".")

sts-ext-value   = 1*(%x21-3A / %x3C / %x3E-7E)
                  ; chars excluding "=", ";", and control chars

The TXT record MUST begin with sts-version field, and the order of
other fields is not significant. If multiple TXT records for "_mta-
sts" are returned by the resolver, records which do not begin with
"v=STSv1;" are discarded. If the number of resulting records is not
one, senders MUST assume the recipient domain does not implement MTA-
STS and skip the remaining steps of policy discovery. If the
resulting TXT record contains multiple strings, then the record MUST
be treated as if those strings are concatenated together without
adding spaces.

3.2. MTA-STS Policies

The policy itself is a set of key/value pairs (similar to [RFC5322]
header fields) served via the HTTPS GET method from the fixed
[RFC5785] "well-known" path of ".well-known/mta-sts.txt" served by
the "mta-sts" host at the Policy Domain. Thus for "example.com" the
path is "https://mta-sts.example.com/.well-known/mta-sts.txt".

When fetching a policy, senders SHOULD validate that the media type
is "text/plain" to guard against cases where webservers allow
untrusted users to host non-text content (typically, HTML or images)
at a user-defined path. All parameters other charset=utf-8 or

charset=us-ascii are ignored. Additional "Content-Type" parameters
are also ignored.

This resource contains the following CRLF-separated key/value pairs:

o "version": (plain-text). Currently only "STSv1" is supported.

o "mode": (plain-text). One of "enforce", "testing", or "none",
   indicating the expected behavior of a sending MTA in the case of a
   policy validation failure. See Section 5, "Policy Application."
   for more details about the three modes.

o "max_age": Max lifetime of the policy (plain-text non-negative
   integer seconds, maximum value of 31557600). Well-behaved clients
   SHOULD cache a policy for up to this value from last policy fetch
   time. To mitigate the risks of attacks at policy refresh time, it
   is expected that this value typically be in the range of weeks or
   greater.

o "mx": MX identity patterns (list of plain-text strings). One or
   more patterns matching a Common Name or Subject Alternative Name
   ([RFC5280]) DNS-ID ([RFC6125]) present in the X.509 certificate
   presented by any MX receiving mail for this domain. For example:

                    mx: mail.example.com <CRLF>
                    mx: .example.net

indicates that mail for this domain might be handled by any MX with a
certificate valid for a host at "mail.example.com" or "example.net".
Valid patterns can be either fully specified names ("example.com") or
suffixes (".example.net") matching the right-hand parts of a server's
identity; the latter case are distinguished by a leading period. If
there are more than one MX specified by the policy, they MUST be on
separate lines within the policy file. In the case of
Internationalized Domain Names ([RFC5891]), the MX MUST specify the
Punycode-encoded A-label [RFC3492] and not the Unicode-encoded
U-label. The full semantics of certificate validation are described
in Section 4.1, "MX Certificate Validation."

An example policy is as below:

                    version: STSv1
                    mode: enforce
                    mx: mail.example.com
                    mx: .example.net
                    mx: backupmx.example.com
                    max_age: 123456

The formal definition of the policy resource, defined using
[RFC7405], is as follows:

```
sts-policy-record          = sts-policy-field *WSP
                             *(CRLF sts-policy-field *WSP)
                             [CRLF]

sts-policy-field           = sts-policy-version /      ; required once
                             sts-policy-mode    /      ; required once
                             sts-policy-max-age /      ; required once

                             0*(sts-policy-mx *WSP CRLF) /
                             ; required at least once, except when
                             ; mode is "none"

                             sts-policy-extension      ; other fields

field-delim                = ":" *WSP

sts-policy-version         = sts-policy-version-field field-delim
                             sts-policy-version-value

sts-policy-version-field = %s"version"

sts-policy-version-value = %s"STSv1"

sts-policy-mode            = sts-policy-mode-field field-delim
                             sts-policy-mode-value

sts-policy-mode-field    = %s"mode"

sts-policy-model-value   =  %s"testing" / %s"enforce" / %s"none"

sts-policy-mx              = sts-policy-mx-field field-delim
                             sts-policy-mx-value

sts-policy-mx-field       = %s"mx"

sts-policy-mx-value       = 1*(ALPHA / DIGIT / "_" / "-" / ".")

sts-policy-max-age         = sts-policy-max-age-field field-delim
                             sts-policy-max-age-value

sts-policy-max-age-field = %s"max_age"

sts-policy-max-age-value = 1*10(DIGIT)

sts-policy-extension       = sts-policy-ext-name   ; additional
```

```
                              field-delim          ; extension
                              sts-policy-ext-value ; fields

   sts-policy-ext-name      = (ALPHA / DIGIT)
                              *31(ALPHA / DIGIT / "_" / "-" / ".")

   sts-policy-ext-value     = 1*(%x21-3A / %x3C / %x3E-7E)
                              ; chars, excluding "=", ";", SP, and
                              ; control chars
```

   Parsers MUST accept TXT records and policy files which are
   syntactically valid (i.e. valid key/value pairs separated by semi-
   colons for TXT records) and but containing additional key/value pairs
   not specified in this document, in which case unknown fields SHALL be
   ignored. If any non-repeated field--i.e. all fields excepting "mx"--
   is duplicated, all entries except for the first SHALL be ignored. If
   any field is not specified, the policy SHALL be treated as invalid.

3.3. HTTPS Policy Fetching

   When fetching a new policy or updating a policy, the HTTPS endpoint
   MUST present a X.509 certificate which is valid for the "mta-sts"
   host (e.g. "mta-sts.example.com") as described below, chain to a
   root CA that is trusted by the sending MTA, and be non-expired. It
   is expected that sending MTAs use a set of trusted CAs similar to
   those in widely deployed Web browsers and operating systems. See
   [RFC5280] for more details about certificate verification.

   The certificate is valid for the "mta-sts" host with respect to the
   rules described in [RFC6125], with the following application-specific
   considerations:

   o Matching is performed only against the DNS-ID identifiers.

   o DNS domain names in server certificates MAY contain the wildcard
      character '*' as the complete left-most label within the
      identifier.

   The certificate MAY be checked for revocation via the Online
   Certificate Status Protocol (OCSP) [RFC6960], certificate revocation
   lists (CRLs), or some other mechanism.

   Policies fetched via HTTPS are only valid if the HTTP response code
   is 200 (OK). HTTP 3xx redirects MUST NOT be followed, and HTTP
   caching (as specified in [RFC7234]) MUST NOT be used.

   Senders may wish to rate-limit the frequency of attempts to fetch the
   HTTPS endpoint even if a valid TXT record for the recipient domain

exists. In the case that the HTTPS GET fails, we suggest
implementations may limit further attempts to a period of five minutes
or longer per version ID, to avoid overwhelming resource-constrained
recipients with cascading failures.

Senders MAY impose a timeout on the HTTPS GET and/or a limit on the
maximum size of the response body to avoid long delays or resource
exhaustion during attempted policy updates. A suggested timeout is
one minute, and a suggested maximum policy size 64 kilobytes; policy
hosts SHOULD respond to requests with a complete policy body within
that timeout and size limit.

If a valid TXT record is found but no policy can be fetched via HTTPS
(for any reason), and there is no valid (non-expired) previously-
cached policy, senders MUST continue with delivery as though the
domain has not implemented MTA-STS.

Conversely, if no "live" policy can be discovered via DNS or fetched
via HTTPS, but a valid (non-expired) policy exists in the sender's
cache, the sender MUST apply that cached policy.

Finally, to mitigate the risk of persistent interference with policy
refresh, as discussed in-depth in Section 10, MTAs SHOULD proactively
refresh cached policies before they expire; a suggested refresh
frequency is once per day. To enable administrators to discover
problems with policy refresh, MTAs SHOULD alert administrators
(through the use of logs or similar) when such attempts fail, unless
the cached policy mode is "none".

3.4. Policy Selection for Smart Hosts and Subdomains

When sending mail via a "smart host"--an administratively configured
intermediate SMTP relay, which is different from the message
recipient's server as determined from DNS --compliant senders MUST
treat the smart host domain as the policy domain for the purposes of
policy discovery and application.

When sending mail to a mailbox at a subdomain, compliant senders MUST
NOT attempt to fetch a policy from the parent zone. Thus for mail
sent to "user@mail.example.com", the policy can be fetched only from
"mail.example.com", not "example.com".

4. Policy Validation

When sending to an MX at a domain for which the sender has a valid
and non-expired MTA-STS policy, a sending MTA honoring MTA-STS MUST
validate:

1. That the recipient MX supports STARTTLS and offers a valid PKIX-
    based TLS certificate.

2. That at least one of the policy's "mx" patterns matches at least
    one of the identities presented in the MX's X.509 certificate, as
    described in "MX Certificate Validation".

This section does not dictate the behavior of sending MTAs when
policies fail to validate; see Section 5, "Policy Application" for a
description of sending MTA behavior when policy validation fails.

4.1. MX Certificate Validation

The certificate presented by the receiving MX MUST chain to a root CA
that is trusted by the sending MTA and be non-expired. The
certificate MUST have a subject alternative name (SAN, [RFC5280])
with a DNS-ID ([RFC6125]) matching the "mx" pattern. The MX's
certificate MAY also be checked for revocation via OCSP [RFC6960],
CRLs [RFC6818], or some other mechanism.

Because the "mx" patterns are not hostnames, however, matching is not
identical to other common cases of X.509 certificate authentication
(as described, for example, in [RFC6125]). Consider the example
policy given above, with an "mx" pattern containing ".example.com".
In this case, if the MX server's X.509 certificate contains a SAN
matching "*.example.com", we are required to implement "wildcard-to-
wildcard" matching.

To simplify this case, we impose the following constraints on
wildcard certificates, identical to those in [RFC7672] section 3.2.3
and [RFC6125] section 6.4.3: wildcards are valid in DNS-IDs, but must
be the entire first label of the identifier (that is,
"*.example.com", not "mail*.example.com"). Senders who are comparing
a "suffix" MX pattern with a wildcard identifier should thus strip
the wildcard and ensure that the two sides match label-by-label,
until all labels of the shorter side (if unequal length) are
consumed.

Note that a wildcard must match a label; an "mx" pattern of
".example.com" thus does not match a SAN of "example.com", nor does a
SAN of "*.example.com" match an "mx" of "example.com".

A simple pseudocode implementation of this algorithm is presented in
Appendix B.

5. **Policy Application**

   When sending to an MX at a domain for which the sender has a valid,
   non-expired MTA-STS policy, a sending MTA honoring MTA-STS applies
   the result of a policy validation failure one of two ways, depending
   on the value of the policy "mode" field:

   1. "enforce": In this mode, sending MTAs MUST NOT deliver the
       message to hosts which fail MX matching or certificate
       validation.

   2. "testing": In this mode, sending MTAs which also implement the
       TLSRPT specification [I-D.ietf-uta-smtp-tlsrpt] merely send a
       report indicating policy application failures (so long as TLSRPT
       is also implemented by the recipient domain).

   3. "none": In this mode, sending MTAs should treat the policy domain
       as though it does not have any active policy; see Section 8.3,
       "Removing MTA-STS", for use of this mode value.

   When a message fails to deliver due to an "enforce" policy, a
   compliant MTA MUST NOT permanently fail to deliver messages before
   checking for the presence of an updated policy at the Policy Domain.
   (In all cases, MTAs SHOULD treat such failures as transient errors
   and retry delivery later.) This allows implementing domains to
   update long-lived policies on the fly.

5.1. **Policy Application Control Flow**

   An example control flow for a compliant sender consists of the
   following steps:

   1. Check for a cached policy whose time-since-fetch has not exceeded
       its "max_age". If none exists, attempt to fetch a new policy
       (perhaps asynchronously, so as not to block message delivery).
       Optionally, sending MTAs may unconditionally check for a new
       policy at this step.

   2. For each candidate MX, in order of MX priority, attempt to
       deliver the message, enforcing STARTTLS and, assuming a policy is
       present, PKIX certificate validation as described in Section 4.1,
       "MX Certificate Validation."

   3. A message delivery MUST NOT be permanently failed until the
       sender has first checked for the presence of a new policy (as
       indicated by the "id" field in the "_mta-sts" TXT record). If a
       new policy is not found, existing rules for the case of temporary

message delivery failures apply (as discussed in [RFC5321]
section 4.5.4.1).

6. Reporting Failures

MTA-STS is intended to be used along with TLSRPT
[I-D.ietf-uta-smtp-tlsrpt] in order to ensure implementing domains
can detect cases of both benign and malicious failures, and to ensure
that failures that indicate an active attack are discoverable. As
such, senders who also implement TLSRPT SHOULD treat the following
events as reportable failures:

o HTTPS policy fetch failures when a valid TXT record is present.

o Policy fetch failures of any kind when a valid policy exists in
   the policy cache, except if that policy's mode is "none".

o Delivery attempts in which a contacted MX does not support
   STARTTLS or does not present a certificate which validates
   according to the applied policy, except if that policy's mode is
   "none".

7. Interoperability Considerations

7.1. SNI Support

To ensure that the server sends the right certificate chain, the SMTP
client MUST have support for the TLS SNI extension [RFC6066]. When
connecting to a HTTP server to retrieve the MTA-STS policy, the SNI
extension MUST contain the name of the policy host (e.g. "mta-
sts.example.com"). When connecting to an SMTP server, the SNI
extension MUST contain the MX hostname.

HTTP servers used to deliver MTA-STS policies MAY rely on SNI to
determine which certificate chain to present to the client. HTTP
servers MUST respond with a certificate chain that matches the policy
hostname or abort the TLS handshake if unable to do so. Clients that
do not send SNI information may not see the expected certificate
chain.

SMTP servers MAY rely on SNI to determine which certificate chain to
present to the client. However servers that have one identity and a
single matching certificate do not require SNI support. Servers MUST
NOT enforce the use of SNI by clients, as the client may be using
unauthenticated opportunistic TLS and may not expect any particular
certificate from the server. If the client sends no SNI extension or
sends an SNI extension for an unsupported server name, the server
MUST simply send a fallback certificate chain of its choice. The

reason for not enforcing strict matching of the requested SNI
hostname is that MTA-STS TLS clients may be typically willing to
accept multiple server names but can only send one name in the SNI
extension. The server's fallback certificate may match a different
name that is acceptable to the client, e.g., the original next-hop
domain.

7.2. Minimum TLS Version Support

MTAs supporting MTA-STS MUST have support for TLS version 1.2
[RFC5246] or higher. The general TLS usage guidance in [RFC7525]
SHOULD be followed.

8. Operational Considerations

8.1. Policy Updates

Updating the policy requires that the owner make changes in two
places: the "_mta-sts" TXT record in the Policy Domain's DNS zone and
at the corresponding HTTPS endpoint. As a result, recipients should
expect a policy will continue to be used by senders until both the
HTTPS and TXT endpoints are updated and the TXT record's TTL has
passed.

In other words, a sender who is unable to successfully deliver a
message while applying a cache of the recipient's now-outdated policy
may be unable to discover that a new policy exists until the DNS TTL
has passed. Recipients should therefore ensure that old policies
continue to work for message delivery during this period of time, or
risk message delays.

Recipients should also prefer to update the HTTPS policy body before
updating the TXT record; this ordering avoids the risk that senders,
seeing a new TXT record, mistakenly cache the old policy from HTTPS.

8.2. Policy Delegation

Domain owners commonly delegate SMTP hosting to a different
organization, such as an ISP or a Web host. In such a case, they may
wish to also delegate the MTA-STS policy to the same organization
which can be accomplished with two changes.

First, the Policy Domain must point the "_mta-sts" record, via CNAME,
to the "_mta-sts" record maintained by the hosting organization.
This allows the hosting organization to control update signaling.

Second, the Policy Domain must point the "well-known" policy location
to the hosting organization. This can be done either by setting the

"mta-sts" record to an IP address or CNAME specified by the hosting
organization and by giving the hosting organization a TLS certificate
which is valid for that host, or by setting up a "reverse proxy"
(also known as a "gateway") server that serves as the Policy Domain's
policy the policy currently served by the hosting organization.

For example, given a user domain "user.example" hosted by a mail
provider "provider.example", the following configuration would allow
policy delegation:

DNS:

     _mta-sts.user.example.   IN CNAME _mta-sts.provider.example.

Policy:

          > GET /.well-known/mta-sts.txt
          > Host: mta-sts.user.example
          < HTTP/1.1 200 OK # Response proxies content from
                             # https://mta-sts.provider.example

Note that while sending MTAs MUST NOT use HTTP caching when fetching
policies via HTTPS, such caching may nonetheless be useful to a
reverse proxy configured as described in this section. An HTTPS
policy endpoint expecting to be proxied for multiple hosted domains--
as with a large mail hosting provider or similar--may wish to
indicate an HTTP Cache-Control "max-age" response directive (as
specified in [RFC7234]) of 60 seconds as a reasonable value to save
reverse proxies an unnecessarily high-rate of proxied policy
fetching.

8.3. Removing MTA-STS

In order to facilitate clean opt-out of MTA-STS by implementing
policy domains, and to distinguish clearly between failures which
indicate attacks and those which indicate such opt-outs, MTA-STS
implements the "none" mode, which allows validated policies to
indicate authoritatively that the policy domain wishes to no longer
implement MTA-STS and may, in the future, remove the MTA-STS TXT and
policy endpoints entirely.

A suggested workflow to implement such an opt out is as follows:

1. Publish a new policy with "mode" equal to "none" and a small
    "max_age" (e.g. one day).

2. Publish a new TXT record to trigger fetching of the new policy.

3. When all previously served policies have expired--normally this
   is the time the previously published policy was last served plus
   that policy's "max_age", but note that older policies may have
   been served with a greater "max_age", allowing overlapping policy
   caches--safely remove the TXT record and HTTPS endpoint.

9. IANA Considerations

9.1. Well-Known URIs Registry

A new "well-known" URI as described in Section 3 will be registered
in the Well-Known URIs registry as described below:

URI Suffix: mta-sts.txt Change Controller: IETF

9.2. MTA-STS TXT Record Fields

IANA is requested to create a new registry titled "MTA-STS TXT Record
Fields". The initial entries in the registry are:

```
+------------+-------------------+-----------------------+
| Field Name |    Description    |       Reference       |
+------------+-------------------+-----------------------+
|     v      |   Record version  | Section 3.1 of RFC XXX |
|     id     | Policy instance ID | Section 3.1 of RFC XXX |
+------------+-------------------+-----------------------+
```

New fields are added to this registry using IANA's "Expert Review"
policy.

9.3. MTA-STS Policy Fields

IANA is requested to create a new registry titled "MTA-STS Policy
Fields". The initial entries in the registry are:

```
+------------+----------------------+-----------------------+
| Field Name |     Description      |       Reference       |
+------------+----------------------+-----------------------+
|  version   |    Policy version    | Section 3.2 of RFC XXX |
|    mode    | Enforcement behavior | Section 3.2 of RFC XXX |
|  max_age   |    Policy lifetime   | Section 3.2 of RFC XXX |
|     mx     |     MX identities    | Section 3.2 of RFC XXX |
+------------+----------------------+-----------------------+
```

New fields are added to this registry using IANA's "Expert Review"
policy.

10. Security Considerations

   SMTP MTA Strict Transport Security attempts to protect against an
   active attacker who wishes to intercept or tamper with mail between
   hosts who support STARTTLS. There are two classes of attacks
   considered:

   o Foiling TLS negotiation, for example by deleting the "250
      STARTTLS" response from a server or altering TLS session
      negotiation. This would result in the SMTP session occurring over
      plaintext, despite both parties supporting TLS.

   o Impersonating the destination mail server, whereby the sender
      might deliver the message to an impostor, who could then monitor
      and/or modify messages despite opportunistic TLS. This
      impersonation could be accomplished by spoofing the DNS MX record
      for the recipient domain, or by redirecting client connections
      intended for the legitimate recipient server (for example, by
      altering BGP routing tables).

   MTA-STS can thwart such attacks only if the sender is able to
   previously obtain and cache a policy for the recipient domain, and
   only if the attacker is unable to obtain a valid certificate that
   complies with that policy. Below, we consider specific attacks on
   this model.

10.1. Obtaining a Signed Certificate

   SMTP MTA-STS relies on certificate validation via PKIX based TLS
   identity checking [RFC6125]. Attackers who are able to obtain a
   valid certificate for the targeted recipient mail service (e.g. by
   compromising a certificate authority) are thus able to circumvent STS
   authentication.

10.2. Preventing Policy Discovery

   Since MTA-STS uses DNS TXT records for policy discovery, an attacker
   who is able to block DNS responses can suppress the discovery of an
   MTA-STS Policy, making the Policy Domain appear not to have an MTA-
   STS Policy. The sender policy cache is designed to resist this
   attack by decreasing the frequency of policy discovery and thus
   reducing the window of vulnerability; it is nonetheless a risk that
   attackers who can predict or induce policy discovery--for example, by
   inducing a sending domain to send mail to a never-before-contacted
   recipient while carrying out a man-in-the-middle attack--may be able
   to foil policy discovery and effectively downgrade the security of
   the message delivery.

Since this attack depends upon intercepting initial policy discovery,
we strongly recommend implementers to prefer policy "max_age" values
to be as long as is practical.

Because this attack is also possible upon refresh of a cached policy,
we suggest implementers do not wait until a cached policy has expired
before checking for an update; if senders attempt to refresh the
cache regularly (for instance, by checking their cached version
string against the TXT record on each successful send, or in a
background task that runs daily or weekly), an attacker would have to
foil policy discovery consistently over the lifetime of a cached
policy to prevent a successful refresh.

Additionally, MTAs should alert administrators to repeated policy
refresh failures long before cached policies expire (through warning
logs or similar applicable mechanisms), allowing administrators to
detect such a persistent attack on policy refresh. (However, they
should not implement such alerts if the cached policy has a "none"
mode, to allow clean MTA-STS removal, as described in Section 8.3.)

Resistance to downgrade attacks of this nature--due to the ability to
authoritatively determine "lack of a record" even for non-
participating recipients--is a feature of DANE, due to its use of
DNSSEC for policy discovery.

10.3. Denial of Service

We additionally consider the Denial of Service risk posed by an
attacker who can modify the DNS records for a recipient domain.
Absent MTA-STS, such an attacker can cause a sending MTA to cache
invalid MX records, but only for however long the sending resolver
caches those records. With MTA-STS, the attacker can additionally
advertise a new, long-"max_age" MTA-STS policy with "mx" constraints
that validate the malicious MX record, causing senders to cache the
policy and refuse to deliver messages once the victim has resecured
the MX records.

This attack is mitigated in part by the ability of a victim domain to
(at any time) publish a new policy updating the cached, malicious
policy, though this does require the victim domain to both obtain a
valid CA-signed certificate and to understand and properly configure
MTA-STS.

Similarly, we consider the possibility of domains that deliberately
allow untrusted users to serve untrusted content on user-specified
subdomains. In some cases (e.g. the service Tumblr.com) this takes
the form of providing HTTPS hosting of user-registered subdomains; in
other cases (e.g. dynamic DNS providers) this takes the form of

   allowing untrusted users to register custom DNS records at the
   provider's domain.

   In these cases, there is a risk that untrusted users would be able to
   serve custom content at the "mta-sts" host, including serving an
   illegitimate MTA-STS policy. We believe this attack is rendered more
   difficult by the need for the attacker to also serve the "_mta-sts"
   TXT record on the same domain--something not, to our knowledge,
   widely provided to untrusted users. This attack is additionally
   mitigated by the aforementioned ability for a victim domain to update
   an invalid policy at any future date.

10.4. Weak Policy Constraints

   Even if an attacker cannot modify a served policy, the potential
   exists for configurations that allow attackers on the same domain to
   receive mail for that domain. For example, an easy configuration
   option when authoring an MTA-STS Policy for "example.com" is to set
   the "mx" equal to ".example.com"; recipient domains must consider in
   this case the risk that any user possessing a valid hostname and CA-
   signed certificate (for example, "dhcp-123.example.com") will, from
   the perspective of MTA-STS Policy validation, be a valid MX host for
   that domain.

10.5. Compromise of the Web PKI System

   A host of risks apply to the PKI system used for certificate
   authentication, both of the "mta-sts" HTTPS host's certificate and
   the SMTP servers' certificates. These risks are broadly applicable
   within the Web PKI ecosystem and are not specific to MTA-STS;
   nonetheless, they deserve some consideration in this context.

   Broadly speaking, attackers may compromise the system by obtaining
   certificates under fraudulent circumstances (i.e. by impersonating
   the legitimate owner of the victim domain), by compromising a
   Certificate Authority or Delegate Authority's private keys, by
   obtaining a legitimate certificate issued to the victim domain, and
   similar.

   One approach commonly employed by Web browsers to help mitigate
   against some of these attacks is to allow for revocation of
   compromised or fraudulent certificates via OCSP [RFC6960] or CRLs
   [RFC6818]. Such mechanisms themselves represent tradeoffs and are
   not universally implemented; we nonetheless recommend implementors of
   MTA-STS to implement revocation mechanisms which are most applicable
   to their implementations.

11. Contributors

Nicolas Lidzborski Google, Inc nlidz (at) google (dot com)

Wei Chuang Google, Inc weihaw (at) google (dot com)

Brandon Long Google, Inc blong (at) google (dot com)

Franck Martin LinkedIn, Inc fmartin (at) linkedin (dot com)

Klaus Umbach 1&1 Mail & Media Development & Technology GmbH
klaus.umbach (at) 1und1 (dot de)

Markus Laber 1&1 Mail & Media Development & Technology GmbH
markus.laber (at) 1und1 (dot de)

12. References

12.1. Normative References

[I-D.ietf-uta-smtp-tlsrpt]
          Margolis, D., Brotman, A., Ramakrishnan, B., Jones, J.,
          and M. Risher, "SMTP TLS Reporting", draft-ietf-uta-smtp-
          tlsrpt-17 (work in progress), March 2018.

[RFC3207] Hoffman, P., "SMTP Service Extension for Secure SMTP over
          Transport Layer Security", RFC 3207, DOI 10.17487/RFC3207,
          February 2002, <https://www.rfc-editor.org/info/rfc3207>.

[RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode
          for Internationalized Domain Names in Applications
          (IDNA)", RFC 3492, DOI 10.17487/RFC3492, March 2003,
          <https://www.rfc-editor.org/info/rfc3492>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
          (TLS) Protocol Version 1.2", RFC 5246,
          DOI 10.17487/RFC5246, August 2008, <https://www.rfc-
          editor.org/info/rfc5246>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S.,
          Housley, R., and W. Polk, "Internet X.509 Public Key
          Infrastructure Certificate and Certificate Revocation List
          (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008,
          <https://www.rfc-editor.org/info/rfc5280>.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321,
          DOI 10.17487/RFC5321, October 2008, <https://www.rfc-
          editor.org/info/rfc5321>.

   [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known
              Uniform Resource Identifiers (URIs)", RFC 5785,
              DOI 10.17487/RFC5785, April 2010, <https://www.rfc-
              editor.org/info/rfc5785>.

   [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS)
              Extensions: Extension Definitions", RFC 6066,
              DOI 10.17487/RFC6066, January 2011, <https://www.rfc-
              editor.org/info/rfc6066>.

   [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and
              Verification of Domain-Based Application Service Identity
              within Internet Public Key Infrastructure Using X.509
              (PKIX) Certificates in the Context of Transport Layer
              Security (TLS)", RFC 6125, DOI 10.17487/RFC6125, March
              2011, <https://www.rfc-editor.org/info/rfc6125>.

   [RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF",
              RFC 7405, DOI 10.17487/RFC7405, December 2014,
              <https://www.rfc-editor.org/info/rfc7405>.

   [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre,
              "Recommendations for Secure Use of Transport Layer
              Security (TLS) and Datagram Transport Layer Security
              (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May
              2015, <https://www.rfc-editor.org/info/rfc7525>.

12.2. Informative References

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119,
              DOI 10.17487/RFC2119, March 1997, <https://www.rfc-
              editor.org/info/rfc2119>.

   [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S.
              Rose, "DNS Security Introduction and Requirements",
              RFC 4033, DOI 10.17487/RFC4033, March 2005,
              <https://www.rfc-editor.org/info/rfc4033>.

   [RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322,
              DOI 10.17487/RFC5322, October 2008, <https://www.rfc-
              editor.org/info/rfc5322>.

   [RFC5891] Klensin, J., "Internationalized Domain Names in
              Applications (IDNA): Protocol", RFC 5891,
              DOI 10.17487/RFC5891, August 2010, <https://www.rfc-
              editor.org/info/rfc5891>.

[RFC6818]   Yee, P., "Updates to the Internet X.509 Public Key
            Infrastructure Certificate and Certificate Revocation List
            (CRL) Profile", RFC 6818, DOI 10.17487/RFC6818, January
            2013, <https://www.rfc-editor.org/info/rfc6818>.

[RFC6960]   Santesson, S., Myers, M., Ankney, R., Malpani, A.,
            Galperin, S., and C. Adams, "X.509 Internet Public Key
            Infrastructure Online Certificate Status Protocol - OCSP",
            RFC 6960, DOI 10.17487/RFC6960, June 2013,
            <https://www.rfc-editor.org/info/rfc6960>.

[RFC7234]   Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke,
            Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching",
            RFC 7234, DOI 10.17487/RFC7234, June 2014,
            <https://www.rfc-editor.org/info/rfc7234>.

[RFC7672]   Dukhovni, V. and W. Hardaker, "SMTP Security via
            Opportunistic DNS-Based Authentication of Named Entities
            (DANE) Transport Layer Security (TLS)", RFC 7672,
            DOI 10.17487/RFC7672, October 2015, <https://www.rfc-
            editor.org/info/rfc7672>.

Appendix A. MTA-STS example record & policy

The owner of "example.com" wishes to begin using MTA-STS with a
policy that will solicit reports from senders without affecting how
the messages are processed, in order to verify the identity of MXs
that handle mail for "example.com", confirm that TLS is correctly
used, and ensure that certificates presented by the recipient MX
validate.

MTA-STS policy indicator TXT RR:

    _mta-sts.example.com. IN TXT "v=STSv1; id=20160831085700Z;"

MTA-STS Policy file served as the response body at "https://mta-
sts.example.com/.well-known/mta-sts.txt":

                    version: STSv1
                    mode: testing
                    mx: mx1.example.com
                    mx: mx2.example.com
                    mx: mx.backup-example.com
                    max_age: 12345678

Appendix B. Message delivery pseudocode

Below is pseudocode demonstrating the logic of a compliant sending
MTA.

While this pseudocode implementation suggests synchronous policy
retrieval in the delivery path, in a working implementation that may
be undesirable, and we expect some implementers to instead prefer a
background fetch that does not block delivery if no cached policy is
present.

```
func isEnforce(policy) {
  // Return true if the policy mode is "enforce".
}

func isNonExpired(policy) {
  // Return true if the policy is not expired.
}

func tryStartTls(connection) {
  // Attempt to open an SMTP connection with STARTTLS with the MX.
}

func isWildcardMatch(pat, host) {
  // Literal matches are true.
  if pat == host {
    return true
  }
  // Leading '.' matches a wildcard against the first part, i.e.
  // .example.com matches x.example.com but not x.y.example.com.
  if pat[0] == '.' {
    parts = SplitN(host, '.', 2)  // Split on the first '.'.
    if len(parts) > 1 && parts[1] == pat[1:] {
      return true
    }
  }
  return false
}

func certMatches(connection, policy) {
  // Assume a handy function to return DNS-ID SANs.
  for san in getDnsIdSansFromCert(connection) {
    for mx in policy.mx {
      // Return if the server certificate from "connection" matches the
      // "mx" host.
      if san[0] == '*' {
        // Invalid wildcard!
```

```
      if san[1] != '.' continue
      san = san[1:]
    }
    if isWildcardMatch(san, mx) || isWildcardMatch(mx, san) {
      return true
    }
    }
  }
  return false
}

func tryDeliverMail(connection, message) {
  // Attempt to deliver "message" via "connection".
}

func tryGetNewPolicy(domain) {
  // Check for an MTA-STS TXT record for "domain" in DNS, and return the
  // indicated policy.
}

func cachePolicy(domain, policy) {
  // Store "policy" as the cached policy for "domain".
}

func tryGetCachedPolicy(domain) {
  // Return a cached policy for "domain".
}

func reportError(error) {
  // Report an error via TLSRPT.
}

func tryMxAccordingTo(message, mx, policy) {
  connection := connect(mx)
  if !connection {
    return false  // Can't connect to the MX so it's not an MTA-STS
                  // error.
  }
  secure := true
  if !tryStartTls(connection) {
    secure = false
    reportError(E_NO_VALID_TLS)
  } else if !certMatches(connection, policy) {
    secure = false
    reportError(E_CERT_MISMATCH)
  }
  if secure || !isEnforce(policy) {
    return tryDeliverMail(connection, message)
```

```
  }
  return false
}

func tryWithPolicy(message, domain, policy) {
  mxes := getMxForDomain(domain)
  for mx in mxes {
    if tryMxAccordingTo(message, mx, policy) {
      return true
    }
  }
  return false
}

func handleMessage(message) {
  domain := ... // domain part after '@' from recipient
  policy := tryGetNewPolicy(domain)
  if policy {
    cachePolicy(domain, policy)
  } else {
    policy = tryGetCachedPolicy(domain)
  }
  if policy {
    return tryWithPolicy(message, domain, policy)
  }
  // Try to deliver the message normally (i.e. without MTA-STS).
}
```

Authors' Addresses

   Daniel Margolis
   Google, Inc

   Email: dmargolis (at) google (dot com)

   Mark Risher
   Google, Inc

   Email: risher (at) google (dot com)

   Binu Ramakrishnan
   Yahoo!, Inc

   Email: rbinu (at) yahoo-inc (dot com)

   Alexander Brotman
   Comcast, Inc

   Email: alex_brotman (at) comcast (dot com)

   Janet Jones
   Microsoft, Inc

   Email: janet.jones (at) microsoft (dot com)