Using TLS in Applications
Internet-Draft
Intended status: Standards Track
Expires: October 5, 2017

D. Margolis
M. Risher
Google, Inc
B. Ramakrishnan
Yahoo!, Inc
A. Brotman
Comcast, Inc
J. Jones
Microsoft, Inc
April 3, 2017

SMTP MTA Strict Transport Security (MTA-STS)
draft-ietf-uta-mta-sts-04

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 5, 2017.

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:

- whether MTAs sending mail to this domain can expect PKIX-authenticated TLS support
- what a conforming client should do with messages when TLS cannot be successfully negotiated

1.1. Terminology

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, SHOULD NOT, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

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

- MTA-STS Policy: A commitment by the Policy Domain to support PKIX authenticated TLS for the specified MX hosts.
- 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 ordinarly be "example.com", but this may be overriden by explicit routing rules (as described in "Policy Selection for Smart Hosts").

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-resistent encrypted tarnsmission. 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 the section "Security Considerations".

In addition, MTA-STS provides an optional report-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 upgrade their 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 "report-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:

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

- "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 [RFC5234], is as follows:

```
sts-text-record = sts-version *WSP %x3B *WSP sts-id [%x3B]

sts-version     = "v" *WSP "=" *WSP %x53 %x54        ; "STSv1"
                  %x53 %x76 %x31

sts-id          = "id" *WSP "=" *WSP 1*32(ALPHA / DIGIT)
```

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.

3.2. MTA-STS Policies

The policy itself is a JSON [RFC4627] object served via the HTTPS GET method from the fixed [RFC5785] "well-known" path of ".well-known/mta-sts.json" 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.json".

This JSON object contains the following key/value pairs:

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

- "mode": (plain-text, required). Either "enforce" or "report", indicating the expected behavior of a sending MTA in the case of a policy validation failure.

- "max_age": Max lifetime of the policy (plain-text non-negative integer seconds, required). 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.

- "mx": MX identity patterns (list of plain-text strings, required). One or more patterns matching a Common Name ([RFC6125]) or Subject Alternative Name ([RFC5280]) DNS-ID present in the X.509 certificate presented by any MX receiving mail for this domain. For example, "["mail.example.com", ".example.net"]" indicates that mail for this domain might be handled by any MX with a certificate valid for a host at "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. 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 "MX Certificate Validation."

An example JSON policy is as below:

```
{
  "version": "STSv1",
  "mode": "enforce",
  "mx": [".mail.example.com"],
  "max_age": 123456
}
```

Parsers SHOULD accept TXT records and policy files which are syntactically valid (i.e. valid key-value pairs separated by semi-colons for TXT records and valid JSON for policy files) and implementing a superset of this specification, in which case unknown fields SHALL be ignored.

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 (as described in [RFC6125]), 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.

HTTP 3xx redirects MUST NOT be followed.

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 implementions 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 to avoid long delays imposed by attempted policy updates. A suggested timeout is one minute; policy hosts SHOULD respond to requests with a complete policy body within that timeout.

If a valid TXT record is found but no policy can be fetched via HTTPS, and there is no valid (non-expired) previously-cached policy, senders MUST treat the recipient domain as one that has not implemented MTA-STS.

3.4. Policy Selection for Smart Hosts and Subdomains

When sending mail via a "smart host"--an intermediate SMTP relay rather than the message recipient's server--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 descriped in "MX Certificate Validation".

This section does not dictate the behavior of sending MTAs when policies fail to validate; in particular, validation failures of policies which specify "report" mode MUST NOT be interpreted as delivery failures, as described in the section "Policy Application".

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 CN-ID ([RFC6125]) or SAN ([RFC5280]) with a DNS-ID matching the "mx" pattern.

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.net". In this case, if the MX server's X.509 certificate contains a SAN matching "*.example.net", 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: wildcards are valid in DNS-IDs or CN-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.

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

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. "report": In this mode, sending MTAs merely send a report (as described in the TLSRPT specification (TODO: add ref)) indicating policy application failures.

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

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.

Finally, in both "enforce" and "report" modes, failures to deliver in compliance with the applied policy result in failure reports to the policy domain, as described in the TLSRPT specification (TODO: add ref).

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, and certificate validation as described in "MX Certificate Validation."

3. Upon message retries, a message MAY be permanently failed following first checking for the presence of a new policy (as indicated by the "id" field in the "_mta-sts" TXT record).

6. Operational Considerations

6.1. 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 thus 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.

7. IANA Considerations

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

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

8. 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:

- 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.

- 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.

8.1. 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.

8.2. 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 victim 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 implementors 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 implementors 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.

Resistence 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.

8.3. Denial of Service

We additionally consider the Denial of Service risk posed by an attacker who can modify the DNS records for a victim 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.

8.4. 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.

9. 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)

10. Appendix 1: 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 JSON served as the response body at [1]

```
{
  "version": "STSv1",
  "mode": "report",
  "mx": ["mx1.example.com", "mx2.example.com"],
  "max_age": 12345678
}
```

11. Appendix 2: 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 implementors 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(mx) {
  // Attempt to open an SMTP connection with STARTTLS with the MX.
}

func certMatches(connection, mx) {
  // For simplicity, we are not checking CNs here.
  for san in getSansFromCert(connection) {
    // Return if the server certificate from "connection" matches the "mx"
host.
    if san[0] == '*' {
      // Invalid wildcard!
      if san[1] != '.' return false
      san = san[1:]
    }
    if san[0] == '.' && HasSuffix(mx, san) {
      return true
    }
    if mx[0] == '.' && HasSuffix(san, mx) {
      return true
    }
    if 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(mx, &connection) {
    secure = false
    reportError(E_NO_VALID_TLS)
  } else if !certMatches(connection, mx) {
    secure = false
    reportError(E_CERT_MISMATCH)
  }
  if secure || !isEnforce(policy) {
    return tryDeliverMail(connection, message)
  }
  return false
}

func tryWithPolicy(message, domain, policy) {
  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, policy)
  }
  // Try to deliver the message normally (i.e. without MTA-STS).
}
```

12. References

12.1. Normative References

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

[RFC3207] Hoffman, P., "SMTP Service Extension for Secure SMTP over Transport Layer Security", RFC 3207, DOI 10.17487/RFC3207, February 2002, <http://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, <http://www.rfc-editor.org/info/rfc3492>.

[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, <http://www.rfc-editor.org/info/rfc4033>.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, DOI 10.17487/RFC4627, July 2006, <http://www.rfc-editor.org/info/rfc4627>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[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, <http://www.rfc-editor.org/info/rfc5280>.

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

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

[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, <http://www.rfc-editor.org/info/rfc6125>.

[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, <http://www.rfc-editor.org/info/rfc7672>.

12.2. URIs

[1] https://mta-sts.example.com/.well-known/mta-sts.json:

Authors' Addresses

Daniel Margolis
Google, Inc

Email: dmargolis (at) google.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.com

Janet Jones
Microsoft, Inc

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