Using TLS in Applications
Internet-Draft
Intended status: Standards Track
Expires: June 18, 2017

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

December 15, 2016

SMTP MTA Strict Transport Security (MTA-STS)
draft-ietf-uta-mta-sts-02

Abstract

SMTP Mail Transfer Agent Strict Transport Security (SMTP STS) is a mechanism enabling mail service providers to declare their ability to receive TLS-secured connections and an expected validity of certificates presented by their MX hosts, 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

Copyright Notice

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 secure mail transmission.

While such _opportunistic_ encryption protocols provide 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 TLS support
- expected validity of server certificates presented by the domain's MX hosts
- 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:

- STS Policy: A committment by the Policy Domain to support PKIX authenticated TLS for the specified MX hosts.
- Policy Domain: The domain for which an STS Policy is defined. (For example, when sending mail to "alice@example.com", the policy domain is "example.com".)
- Policy Authentication: Authentication of the STS policy retrieved for a recipient domain by the sender.

2. Related Technologies

The DANE TLSA record [RFC7672] is similar, in that DANE is also designed to upgrade opportunistic, unauthenticated encryption into required, authenticated encryption. DANE requires DNSSEC [RFC4033] for authentication; the mechanism described here instead relies on certificate authorities (CAs) and does not require DNSSEC. For a thorough discussion of this trade-off, see the section _Security_ _Considerations_.

In addition, SMTP STS provides an optional report-only mode, enabling soft deployments to detect policy failures.

3. Policy Discovery

SMTP 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 patterns (list of plain-text MX match strings, required). One or more patterns matching the expected MX for this domain. For example, "["*.example.com", "*.example.net"]" indicates that mail for this domain might be handled by any MX with a hostname at "example.com" or "example.net". Valid patterns can be either hostname literals (e.g. "mx1.example.com") or wildcard matches, so long as the wildcard occupies the full left-most label in the pattern. (Thus "*.example.com" is valid but "mx*.example.com" is not.)

An example JSON policy is as below:

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

A lenient parser 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 TLS 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.

3.4. Policy Selection for Smart Hosts

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.

4. Policy Validation

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

1. That the recipient MX matches the "mx" pattern from the recipient domain's policy.

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

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 Matching

When delivering mail for the Policy Domain to a recipient MX host, the sender validates the MX match against the "mx" pattern from the applied policy. The semantics for these patterns are those found in section 6.4 of [RFC6125].

Patterns may contain a wildcard character "*" which matches any single domain name component or component fragment, though only as the leftmost component in a pattern. For example, "*.example.com" is a valid pattern, but "foo.*.example.com" is not. Given the pattern "*.example.com", "mx1.example.com" is a valid MX host, but "1234.dhcp.example.com" is not.

4.2. MX Certificate Validation

The certificate presented by the receiving MX MUST be valid for the MX hostname and chain to a root CA that is trusted by the sending MTA. The certificate MUST have a CN or SAN matching the MX hostname (as described in [RFC6125]) and be non-expired.

In the case of an "implicit" MX record (as specified in [RFC2821]) where no MX RR exists for the recipient domain but there is an A RR, the MX hostname is assumed to be that of the A RR and should be validated as such.

5. Policy Application

When sending to an MX at a domain for which the sender has a valid, non-expired STS policy, a sending MTA honoring SMTP STS applies the result of a policy validation 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 treat STS policy failures as a mail delivery error, and MUST NOT deliver the message to this host.

When a message fails to deliver due to an "enforce" policy, a compliant MTA MUST check for the presence of an updated policy at the Policy Domain before permanently failing to deliver the message.

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. MX Preference

When applying a policy, sending MTAs SHOULD select recipient MXs by first eliminating any MXs at lower priority than the current host (if in the MX candidate set), then eliminating any non-matching (as specified by the STS Policy) MX hosts from the candidate MX set, and then attempting delivery to matching hosts as indicated by their MX priority, until delivery succeeds or the MX candidate set is empty.

5.2. 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. (Optionally, sending MTAs may unconditionally check for a new policy at this step.)

2. Filter candidate MXs against the current policy.

3. If no candidate MXs are valid and the policy mode is "enforce", temporarily fail the message. (Otherwise, generate a failure report but deliver as though MTA STS were not implemented.)

4. For each candidate MX, in order of MX priority, attempt to deliver the message, enforcing STARTTLS and the MX host's PKIX certificate validation.

5. 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. In the case where the HTTPS

endpoint has been updated but the TXT record has not yet been, senders will not know there is a new policy released and may thus continue to use old, previously cached versions. 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.

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 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:

1. 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.

2. 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).

SMTP Strict Transport Security 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.

Since we use DNS TXT records for policy discovery, an attacker who is able to block DNS responses can suppress the discovery of an STS Policy, making the Policy Domain appear not to have an STS Policy. The sender policy cache is designed to resist this attack.

We additionally consider the Denial of Service risk posed by an attacker who can modify the DNS records for a victim domain. Absent SMTP STS, such an attacker can cause a sending MTA to cache invalid MX records for a long TTL. With SMTP STS, the attacker can

additionally advertise a new, long-"max_age" SMTP 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 SMTP STS.

Similarly, we consider the possibilty 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 SMTP STS policy. We believe this attack is rendered more difficult by the need for the attacker to both inject malicious (but temporarily working) MX records and 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.

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 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 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: Domain Owner STS example record

10.1. Example 1

The owner of "example.com" wishes to begin using STS with a policy that will solicit reports from receivers 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.

STS policy indicator TXT RR:

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

STS Policy JSON served as the response body at [1]

```
{
  "version": "STSv1",
  "mode": "report",
  "mx": ["mx1.example.com", "mx2.example.com"],
  "max_age": 123456
}
```

11. Appendix 2: Message delivery pseudocode

Below is pseudocode demonstrating the logic of a complaint sending MTA. This implements the "two-pass" approach, first attempting delivery with a newly fetched policy (if present) before falling back to a cached policy (if 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) {
  // Return if the server certificate from "connection" matches the "mx" host.
}

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

func getMxsForPolicy(domain, policy) {
  // Sort the MXs by priority, filtering out those which are invalid according
  // to "policy".
}

func tryGetNewPolicy(domain) {
  // Check for an MTA STS TXT record for "domain" in DNS, and return the
  // indicated policy (or a local cache of the unvalidated policy).
}

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

func tryGetCachedPolicy(domain, policy) {
  // 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 STS error.
  }
  status := !(tryStartTls(mx, &connection) && certMatches(connection, mx))
  status = true
  if !tryStartTls(mx, &connection) {
    status = false
    reportError(E_NO_VALID_TLS)
  } else if certMatches(connection, mx) {
    status = false
    reportError(E_CERT_MISMATCH)
  }
  if status || !isEnforce(policy) {
    return tryDeliverMail(connection, message)
```

```
  }
  return false
}

func tryWithPolicy(message, domain, policy) {
  mxes := getMxesForPolicy(domain, policy)
  if mxs is empty {
    reportError(E_NO_VALID_MXES)
  }
  for mx in mxes {
    if tryMxAccordingTo(message, mx, policy) {
      return true
    }
  }
  return false
}

func handleMessage(message) {
  domain := ... // domain part after '@' from recipient
  oldPolicy := tryGetCachedPolicy(domain)
  newPolicy := tryGetNewPolicy(domain)
  if newPolicy {
    cachePolicy(domain, newPolicy)
    oldPolicy = newPolicy
  }
  if oldPolicy {
    return tryWithPolicy(message, oldPolicy)
  }
  // There is no policy or there's a new policy that did not work.
  // Try to deliver the message normally (i.e. without 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>.

[RFC2821] Klensin, J., Ed., "Simple Mail Transfer Protocol", RFC 2821, DOI 10.17487/RFC2821, April 2001, <http://www.rfc-editor.org/info/rfc2821>.

[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>.

[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>.

[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>.

[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: alexander_brotman (at) cable.comcast (dot com)

Janet Jones
Microsoft, Inc

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