TLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 30, 2013

M. Marlinspike
T. Perrin, Ed.

September 26, 2012

Trust Assertions for Certificate Keys
draft-perrin-tls-tack-01.txt

Abstract

This document defines TACK, a TLS Extension that enables a TLS server to assert the authenticity of its public key. A "tack" contains a "TACK key" which is used to sign the public key from the TLS server's certificate. Hostnames can be "pinned" to a TACK key. TLS connections to a pinned hostname require the server to present a tack containing the pinned key and a corresponding signature over the TLS server's public key.

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 March 30, 2013.

Copyright Notice

Table of Contents

1. Introduction

Traditionally, a TLS client verifies a TLS server's public key using a certificate chain issued by some public CA. "Pinning" is a way for clients to obtain increased certainty in server public keys. Clients that employ pinning check for some constant "pinned" element of the TLS connection when contacting a particular TLS host.

Unfortunately, a number of problems arise when attempting to pin certificate chains: the TLS servers at a given hostname may have different certificate chains simultaneously deployed and may change their chains at any time, the "more constant" elements of a chain (the CAs) may not be trustworthy, and the client may be oblivious to key compromise events which render the pinned data untrustworthy.

TACK addresses these problems by having the site sign its TLS server public keys with a "TACK key". This enables clients to "pin" a hostname to the TACK key without requiring sites to modify their existing certificate chains, and without limiting a site's flexibility to deploy different certificate chains on different servers or change certificate chains at any time. Since TACK pins are based on TACK keys (instead of CA keys), trust in CAs is not required. Additionally, the TACK key may be used to revoke compromised TLS private keys, and TACK key rollovers may be performed to recover from suspect or compromised TACK keys.

If requested, a compliant server will send a TLS Extension containing its "tack". Inside the tack is a public key and signature. Once a client has seen the same (hostname, TACK public key) pair multiple times, the client will "activate" a pin between the hostname and TACK key for a period equal to the length of time the pair has been observed for. This "pin activation" algorithm limits the impact of bad pins resulting from transient network attacks or operator error.

TACK pins are easily shared between clients. For example, a TACK client may scan the internet to discover TACK pins, then publish

these pins through some 3rd-party trust infrastructure for other clients to rely upon.

2. Requirements notation

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].

3. Overview

3.1. Tack life cycle

A server operator using TACK may perform several processes:

Selection of a TACK key: The server operator first chooses the ECDSA signing key to use for a set of hostnames. It is safest to use a different signing key for each hostname, though a signing key may be reused for closely-related hostnames (such as aliases for the same host, or hosts sharing the same TLS key).

Creating initial tacks under a TACK key: The TACK private key is then used to sign the TLS public keys for all servers associated with those hostnames. The TACK public key and signature are combined with some metadata into each server's "tack".

Deploying initial tacks: For each hostname, tacks are deployed to TLS servers in a two-stage process. First, each TLS server associated with the hostname is given a tack. Once this is completed, the tacks are activated by setting the "activation flag" on each server.

Creating new tacks under a TACK key: A tack needs to be replaced whenever a server changes its TLS public key, or when the tack expires. Tacks may also need to be replaced with later-generation tacks if the TACK key's "min_generation" is updated (see next).

Revoking old tacks: If a TLS private key is compromised, the tacks signing this key can be revoked by publishing a new tack containing a higher "min_generation".

Deactivating tacks: If a server operator wishes to stop deploying tacks, all tacks for a hostname can be deactivated via the activation flag, allowing the server to remove the tacks within 30 days (at most).

Rollover: If a server operator wishes to change the TACK key a hostname is pinned to, the server can publish a new tack alongside the old one. This lets clients activate pins for the new TACK key prior to the server deactivating the older pins.

3.2. Pin life cycle

A TACK pin associates a hostname and a TACK key. Pins are grouped into "pin stores". A client may populate its pin stores by either performing "pin activation" directly, or by querying some other party. For example, a client application may have a store for pin activation as well as a store whose contents are periodically fetched from a server.

Whenever a client performing "pin activation" sees a hostname and TACK key combination not represented in the "pin activation" pin store, an inactive pin is created. Every subsequent time the client sees the same pin, the pin is "activated" for a period equal to the timespan between the first time the pin was seen and the most recent time, up to a maximum period of 30 days.

A pin store may contain up to two pins per hostname. This allows for "pin rollover", where a server securely transitions from one pin to another. If both pins are simultaneously active, then the server must satisfy both of them by presenting a pair of tacks.

In addition to creating and activating pins, a TLS connection can alter client pin stores by publishing new "min_generation" values in a tack. Each pin stores the highest "min_generation" value it has seen from the pinned TACK key, and rejects tacks from earlier generations.

4. TACK Extension

4.1. Definition of TackExtension

A new TLS ExtensionType ("tack") is defined and MAY be included by a TLS client in the ClientHello message defined in [RFC5246].

```
enum {tack(TBD), (65535)} ExtensionType;
```

The "extension_data" field of this ClientHello extension SHALL be empty. A TLS server which is not resuming a TLS session MAY respond with an extension of type "tack" in the ServerHello. The "extension_data" field of this ServerHello extension SHALL contain a "TackExtension", as defined below using the TLS presentation language from [RFC5246].

```
struct {
   opaque public_key[64];
   uint8  min_generation;
   uint8  generation;
   uint32 expiration;
   opaque target_hash[32];
   opaque signature[64];
} Tack;  /* 166 bytes */

struct {
   Tack   tacks<166...332>   /* 1 or 2 Tacks */
   uint8  activation_flags;  /* 0...3 */
} TackExtension;
```


4.2. Explanation of TackExtension fields

4.2.1. Tack fields

public_key: This field specifies the tack's public key. The field contains a pair of integers (x, y) representing a point on the elliptic curve P-256 defined in [FIPS186-3]. Each integer is encoded as a 32-byte octet string using the Integer-to-Octet-String algorithm from [RFC6090], and these strings are concatenated with the x value first. (NOTE: This is equivalent to an uncompressed subjectPublicKey from [RFC5480], except that the initial 0x04 byte is omitted).

min_generation: This field publishes a min_generation value.

generation: This field assigns each tack a generation. Generations less than a published min_generation are considered revoked.

expiration: This field specifies a time after which the tack is considered expired. The time is encoded as the number of minutes, excluding leap seconds, after midnight UTC, January 1 1970.

target_hash: This field is a hash of the TLS server's SubjectPublicKeyInfo [RFC5280] using the SHA256 algorithm from [FIPS180-2]. The SubjectPublicKeyInfo is typically conveyed as part of the server's X.509 certificate.

signature: This field is an ECDSA signature by the tack's public key over the 8 byte ASCII string "tack_sig" followed by the contents of the tack prior to the "signature" field (i.e. the preceding 102 bytes). The field contains a pair of integers (r, s) representing an ECDSA signature as defined in [FIPS186-3], using curve P-256 and SHA256. Each integer is encoded as a 32-byte octet string using the Integer-to-Octet-String algorithm from [RFC6090], and these strings are concatenated with the r value first.

4.2.2. TackExtension fields

tacks: This field provides the server's tack(s). It SHALL contain 1 or 2 tacks.

activation_flags: This field contains "activation flags" for the extension's tacks. If the low order bit is set, the first tack is considered active. If the next lowest bit is set, the second tack is considered active. An active tack MAY be used by the pin activation algorithm in Section 5.3.4 to create, activate, and extend the activation of TACK pins.

5. Client processing

5.1. TACK pins

A client SHALL have a local store of pins, and MAY have multiple stores. Each pin store consists of a map associating fully qualified DNS hostnames with either one or two sets of the following values:

Initial time: A timestamp noting when this pin was created.

End time: A timestamp determining the pin's "active period". If set to zero or a time in the past, the pin is "inactive". If set to a future time, the pin is "active" until that time.

TACK public key (or hash): A public key or a cryptographically-secure, second preimage-resistant hash of a public key.

Min_generation: A single byte used to detect revoked tacks. All pins within a pin store sharing the same TACK public key SHALL have the same min_generation.

A hostname along with the above values comprises a "TACK pin". Thus, each store can hold up to two pins for a hostname (however, those two pins MUST reference different public keys). A pin "matches" a tack if they reference the same public key. A pin is "relevant" if its hostname equals the TLS server's hostname.

5.2. High-level client processing

A TACK client SHALL send the "tack" extension defined previously, and SHALL send the "server_name" extension from [RFC6066]. If not resuming a session, the server MAY respond with a TackExtension. Regardless of whether a TackExtension is returned, the client SHALL perform the following steps prior to using the connection:

1. Check whether the TLS handshake is "well-formed".

2. For each pin store, do:

    A. Check tack generations and update min_generations.

    B. Determine the store's status.

    C. Perform pin activation (optional).

These steps SHALL be performed in order. If there is any error, the client SHALL send a fatal error alert and close the connection, skipping the remaining steps (see Section 5.3 for details).

Based on step 2B, each store will report one of three statuses for the connection: "accepted", "rejected", or "unpinned". A rejected connection might indicate a network attack. If the connection is rejected the client SHOULD send a fatal "access_denied" error alert and close the connection.

A client MAY perform additional verification steps before using an accepted or unpinned connection. See Section 6.3 for an example.

5.3. Client processing details

5.3.1. Check whether the TLS handshake is well-formed

A TLS handshake is "well-formed" if the following are true. Unless otherwise specified, if any of the following are false a "bad_certificate" fatal error alert SHALL be sent.

1. The handshake protocol negotiates a cryptographically secure ciphersuite and finishes succesfully.

2. If a TackExtension is present then all length fields are correct, "activation_flags" is <= 3, and the tacks are "well-formed" (see below).

3. If there are two tacks, they have different "public_key" fields.

A tack is "well-formed" if:

1. "generation" is >= "min_generation".

2. "expiration" specifies a time in the future, otherwise the client SHALL send a fatal "certificate_expired" error alert.

3. "target_hash" is a correct hash of the SubjectPublicKeyInfo.

4. "signature" is a correct ECDSA signature.

5.3.2. Check tack generations and update min_generations

If a tack has matching pins in the pin store and a generation less than the stored min_generation, then that tack is revoked and the client SHALL send a fatal "certificate_revoked" error alert. If a tack has matching pins and a min_generation greater than the stored min_generation, the stored value SHALL be set to the tack's value.

5.3.3. Determine the store's status

If there is a relevant active pin without a matching tack, then the connection is "rejected". If the connection is not rejected and there is a relevant active pin with a matching tack, then the connection is "accepted". Otherwise, the connection is "unpinned".

5.3.4. Pin activation (optional)

The TLS connection MAY be used to create, delete, and activate pins. This "pin activation algorithm" is optional; a client MAY rely on an external source of pins. If the connection was "rejected" by the previous processing step, then pin activation is skipped.

The first step in pin activation is to evaluate each relevant pin (there may be one or two):

1. If a pin has no matching tack, its handling will depend on whether the pin is active. If active, the connection will have been rejected, skipping pin activation. If inactive, the pin SHALL be deleted, since it is contradicted by the connection.

2. If a pin has a matching tack, its handling will depend on whether the tack is active. If inactive, the pin is left unchanged. If active, the pin SHALL have its "end time" set based on the current, initial, and end times:

   end = current + MIN(30 days, current - initial)

In sum: (1) deletes unmatched pins, provided they are inactive; and (2) activates matched pins, provided the matching tack is active.

The remaining step in pin activation is to add new inactive pins for any unmatched active tacks. Each new pin uses the server's hostname, the tack's public key and min_generation (unless the store has a higher min_generation for the public key), an "initial time" set to the current time, and an "end time" of zero.

(Note that there are always sufficient empty "slots" in the pin store for adding new pins without exceeding two pins per hostname. This is because the number of matching pins equals the number of matching tacks, so the number of empty pin slots equals the number of unmatched tacks.)

The following tables summarize this behavior from the perspective of a pin. You can follow the lifecycle of a single pin from "New inactive pin" to "Delete pin".

Relevant pin is active:

Pin matches a tack	Tack is active	Result
Yes	Yes	Extend activation period
Yes	No	-
No	-	(Connection rejected)

Relevant pin is inactive:

Pin matches a tack	Tack is active	Result
Yes	Yes	Activate pin
Yes	No	-
No	-	Delete pin

Tack doesn't match any relevant pin:

Unmatched tack is active	Result
Yes	New inactive pin
No	-

6. Application protocols and TACK

6.1. Pin scope

TACK pins are specific to a particular application protocol. In other words, a pin for HTTPS at "example.com" implies nothing about POP3 or SMTP at "example.com".

6.2. TLS negotiation

Some application protocols negotiate TLS as an optional feature (e.g. SMTP using STARTTLS [RFC3207]). If such a server fails to negotiate TLS and there are relevant active pins, then the connection is rejected by the pin. If the server fails to negotiate TLS, then any relevant, inactive pins SHALL be deleted. Note that these steps are taken despite the absence of a TLS connection.

6.3. Certificate verification

A TACK client MAY choose to perform some form of certificate verification in addition to TACK processing. When combining certificate verification and TACK processing, the TACK processing described in Section 5 SHALL be followed, with the exception that TACK processing MAY be terminated early (or skipped) if some fatal certificate error is discovered.

If TACK processing and certificate verification both complete without a fatal error, the client SHALL apply some policy to decide whether to accept the connection. The policy is up to the client. An example policy would be to accept the connection only if it passes certificate verification and is not rejected by a pin.

7. Fingerprints

A "key fingerprint" may be used to represent a TACK public key to users in a form that is easy to compare and transcribe. A key fingerprint consists of the first 25 characters from the base32 encoding of SHA256(public_key), split into 5 groups of 5 characters separated by periods. Base32 encoding is as specified in [RFC4648], except lowercase is used. Examples:

g5p5x.ov4vi.dgsjv.wxctt.c5iul

quxiz.kpldu.uuedc.j5znm.7mqst

e25zs.cth7k.tscmp.5hxdp.wf47j

8. Advice

8.1. For server operators

Key reuse: All servers that are pinned to a single TACK key are able to impersonate each other, since clients will perceive their tacks as equivalent. Thus, TACK keys SHOULD NOT be reused with different hostnames unless these hostnames are closely related. Examples where it would be safe to reuse a TACK key are hostnames aliased to the same host, hosts sharing the same TLS key, or hostnames for a group of near-identical servers.

Aliases: A TLS server may be referenced by multiple hostnames. Clients may pin any of these hostnames. Server operators should be careful when using DNS aliases that hostnames are not pinned inadvertently.

Generations: To revoke older generations of tacks, the server operator SHOULD first provide all servers with a new generation of tacks, and only then provide servers with new tacks containing the new min_generation. Otherwise, a client may receive a min_generation update from one server but then try to contact an older-generation server which has not yet been updated.

Tack expiration: When TACK is used in conjunction with certificates it is recommended to set the tack expiration equal to the end-entity certificate expiration plus 30 days, allowing the tack and certificate to both be replaced at the same time. The extra 30 days ensures there is enough time to employ "pin deactivation" (see below) if the TACK private key is lost. Alternatively, short-lived tacks may be used so that a compromised TLS private key has limited value to an attacker.

Tack/pin activation: Tacks should only be activated once all TLS servers sharing the same hostname have a tack. Otherwise, a client may activate a pin by contacting one server, then contact a different server at the same hostname that does not yet have a tack.

Tack/pin deactivation: If all servers at a hostname deactivate their tacks (by clearing the activation flags), all existing pins for the hostname will eventually become inactive. The tacks can be removed after a time interval equal to the maximum active period of any affected pins (30 days at most).

Pin rollover: When performing a rollover, the old and new tacks SHOULD be published simultaneously for at least 60 days. This ensures that a pin activation client who is contacting the server at least once every 30 days will not have the length of its activation periods affected by the transition. Example rollover process: Add new tacks; activate new tacks; wait 30+ days; deactivate old tacks; wait 30+ days; remove old tacks.

8.2. For client implementers

Sharing pin information: It is possible for a client to maintain a pin store based entirely on its own TLS connections. However, such a client runs the risk of creating incorrect pins, failing to keep its pins active, or failing to receive min_generation updates. Clients are advised to make use of 3rd-party trust infrastructure so that pin data can be aggregated and shared. This will require additional protocols outside the scope of this document.

Clock synchronization: A client SHOULD take measures to prevent tacks from being erroneously rejected as expired due to an inaccurate client clock. Such methods MAY include using time synchronization protocols such as NTP [RFC5905], or accepting seemingly-expired tacks as "well-formed" if they expired less than T minutes ago, where T is a "tolerance bound" set to the client's maximum expected clock error.

9. Security considerations

9.1. For server operators

All servers pinned to the same TACK key can impersonate each other (see Section 8.1). Think carefully about this risk if using the same TACK key for multiple hostnames.

Make backup copies of the TACK private key and keep all copies in secure locations where they can't be compromised.

A TACK private key MUST NOT be used to perform any non-TACK cryptographic operations. For example, using a TACK key for email encryption, code-signing, or any other purpose MUST NOT be done.

HTTP cookies [RFC6265] set by a pinned host can be stolen by a network attacker who can forge web and DNS responses so as to cause a client to send the cookies to a phony subdomain of the pinned host. To prevent this, TACK HTTPS Servers SHOULD set the "secure" attribute and omit the "domain" attribute on all security-sensitive cookies, such as session cookies. These settings tell the browser that the cookie should only be presented back to the originating host (not its subdomains), and should only be sent over HTTPS (not HTTP) [RFC6265].

9.2. For client implementers

A TACK pin store may contain private details of the client's connection history. An attacker may be able to access this information by hacking or stealing the client. Some information about the client's connection history could also be gleaned by observing whether the client accepts or rejects connections to phony TLS servers without correct tacks. To mitigate these risks, a TACK client SHOULD allow the user to edit or clear the pin store.

Aside from rejecting TLS connections, clients SHOULD NOT take any actions which would reveal to a network observer the state of the client's pin store, as this would allow an attacker to know in advance whether a "man-in-the-middle" attack on a particular TLS connection will succeed or be detected.

An attacker may attempt to flood a client with spurious tacks for different hostnames, causing the client to delete old pins to make space for new ones. To defend against this, clients SHOULD NOT delete active pins to make space for new pins. Clients instead SHOULD delete inactive pins. If there are no inactive pins to delete, then the pin store is full and there is no space for new pins. To select an inactive pin for deletion, the client SHOULD delete the pin with the oldest "end time".

9.3. Note on algorithm agility

If the need arises for tacks using different cryptographic algorithms (e.g., if SHA256 or ECDSA are shown to be weak), a "v2" version of tacks could be defined, requiring assignment of a new TLS Extension number. Tacks as defined in this document would then be known as "v1" tacks.

10. IANA considerations

10.1. New entry for the TLS ExtensionType Registry

IANA is requested to add an entry to the existing TLS ExtensionType registry, defined in [RFC5246], for "tack"(TBD) as defined in this document.

11. Acknowledgements

Valuable feedback has been provided by Adam Langley, Chris Palmer, Nate Lawson, and Joseph Bonneau.

12. Normative references

[FIPS180-2]
National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-2, August 2002, <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>.

[FIPS186-3]
National Institute of Standards and Technology, "Digital Signature Standard", FIPS PUB 186-3, June 2009, <http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf>.

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

[RFC3207] Hoffman, P., "SMTP Service Extension for Secure SMTP over Transport Layer Security", RFC 3207, February 2002.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[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, May 2008.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, March 2009.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

Authors' Addresses

Moxie Marlinspike

Trevor Perrin (editor)

Email: tack@trevp.net