TLS Working Group M. Marlinspike
Internet-Draft T. Perrin, Ed.
Intended status: Standards Track May 22, 2012
Expires: November 23, 2012

Trust Assertions for Certificate Keys
draft-perrin-tls-tack-00.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.

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 previous TACK signatures (or even itself) in order to handle the compromise of TLS or TACK private 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" process 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 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, pin activation is enabled on the servers.

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".

Revoking TACK keys: If a TACK private key is compromised, or a server operator wishes to stop using TACK or abruptly change its TACK key for any reason, a server can revoke an entire TACK key (including all TACKs and pins referring to it) by publishing a "break signature".

3.2. Pin life cycle

A TACK client maintains a store of pins for verifying TLS connections. Pins associate a hostname and a TACK key. When a client sees a new hostname and TACK key combination, 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.

Pin activation prevents an attacker with short-lived control of the hostname from activating long-lived pins. It also makes it safer for sites to experiment with TACKs, as a new TACK can be discarded without causing long-lived problems. The 30 day limit guarantees that a worst-case pin can be recovered from in reasonable time.

In addition to creating and activating pins, a TLS connection can alter the clients's pin store by publishing revocation data:

Min_generation: Each pin stores the highest "min_generation" value it has seen from the pinned TACK key, and rejects TACKs from earlier generations.

Break signatures: A TLS handshake may send break signatures which cause all pins for the broken key to be discarded.

4. TACK Extension

4.1. Definition of TACK_Extension

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 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 SHALL contain a "TACK_Extension", as defined below using the TLS presentation language from [RFC5246].

```
enum (disabled(0), enabled(1)} TACK_Activation;

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

struct {
   opaque public_key[64];
   opaque signature[64];
} TACK_Break_Sig;  /* 128 bytes */

struct {
   TACK tack<0...166>  /* 0 or 1 TACK */
   TACK_Break_Sig  break_sigs<0...1024>  /* 0...8 Break Sigs */
   TACK_Activation pin_activation;
} TACK_Extension;
```

4.2. Explanation of TACK_Extension 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. TACK_Break_Sig fields

public_key: This field specifies the TACK key being broken. The key is encoded as per TACK.public_key.

signature: This field is an ECDSA signature by the TACK_Break_Sig's public key over the 14 byte ASCII string "tack_break_sig". The field contains a pair of integers (r, s) representing an ECDSA signature as defined in [FIPS186-3], using curve P-256 and SHA256. It is calculated and encoded as per TACK.signature.

4.2.3. TACK_Extension fields

tack: This field provides the server's TACK. It MAY be empty, or MAY contain a TACK.

break_sigs: This field provides break signatures. It MAY be empty, or MAY contain up to 8 break signatures.

pin_activation: If pin activation is enabled, then the TACK_Extension MAY be used by clients to activate or extend the activation of TACK pins. This field is typically toggled from a disabled to an enabled state once TACKs have been deployed to all TLS servers for a hostname. Enabling pin_activation when there is no TACK present has no effect.

Note that both the "tack" and "break_sigs" fields MAY be empty.

5. Client processing

5.1. TACK pins, key records, and name records

A client supporting TACK SHALL have a local store of pins, consisting of "key records" and "name records". Each name record is associated with a key record. Multiple name records MAY be associated with one key record. A "pin" refers to a (name record, key record) pair.

A "key record" contains:

TACK public key (or hash): A public key or a cryptographically-secure, second preimage-resistant hash of a public key. A client SHALL NOT store multiple key records referencing the same key.

Min_generation: A single byte used to detect revoked TACKs.

A "name record" contains:

Name: A fully qualified DNS hostname. A client SHALL NOT store multiple name records with the same name. The TLS server's hostname is considered the "relevant name", and a pin whose name exactly matches the relevant name is considered a "relevant pin".

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

Active period end: Empty or a timestamp. If empty or set to a time in the past, the pin is "inactive". If set to a future time, the pin is "active" until that time.

5.2. High-level client processing

A TACK client SHALL send the "tack" extension defined previously, and SHOULD send the "server_name" extension from [RFC6066]. If not resuming a session, the server MAY respond with a TACK_Extension. A TACK client SHALL perform the following steps prior to using a non-resumed connection:

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

2. Check the TACK generation and update min_generation.

3. Check whether the TACK is expired.

4. Create and activate pins (optional).

5. Discard pins based on break signatures.

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).

After the above steps, if there is a relevant active pin and a TACK whose key is referenced by the pin, then the connection is "accepted" by the pin. If there is a relevant active pin but no such TACK, the connection is "rejected" by the pin. If there is no relevant active pin, the connection is "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 6.1 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 (the error alert to be sent on a failure is indicated in parentheses):

1. The handshake protocol negotiates a cryptographically secure ciphersuite and finishes succesfully (else see [RFC5246]).

2. The handshake contains either no TACK_Extension or a syntactically-valid TACK_Extension (else "decode_error").

3. If break signatures are present, the signatures are correct (else "decrypt_error"). This step is optional, as break signature verification MAY be deferred till later.

4. If a TACK is present, it is "well-formed" by the rules below.

A TACK is "well-formed" if:

1. "public_key" is a valid elliptic curve public key on the curve P-256 (else "decrypt_error").

2. "generation" is >= "min_generation" (else "decode_error").

3. "target_hash" is equal to the SHA256 hash of the server's SubjectPublicKeyInfo (else "illegal_parameter").

4. "signature" is a correct ECDSA signature (else "decrypt_error").

5.3.2. Check the TACK generation and update min_generation

If there is a TACK and a key record referencing the TACK key, and the TACK's generation is less than the key record's min_generation, then the TACK is revoked and the client SHALL send the "certificate_revoked" alert and close the connection.

Otherwise, if there is a TACK and a key record referencing the TACK key, and the TACK's min_generation is greater than the key record's min_generation, then the key record's min_generation SHALL be set to the TACK's value.

5.3.3. Check whether the TACK is expired

If there is a TACK and the TACK's "expiration" field specifies a time in the past, the client SHALL send the "certificate_expired" alert and close the connection.

5.3.4. Create and activate pins (optional)

The TLS connection MAY be used to create, delete, and activate pins as described in this section. Note that this section is optional; a client MAY rely on an external source of pins, provided the external pins are produced by a client following the below algorithms.

If there is a TACK and a relevant pin referencing the TACK key, and pin activation is enabled, the name record's "active period end" SHALL be set using the below formula (where "current" is the current time, and "initial" is the "initial timestamp" from the name record):

```
active_period_end = current + MIN(30 days, current - initial)
```

If there is a TACK and either no relevant pin or an inactive relevant pin that does not reference the TACK key, a new pin SHALL be created:

1. If the TACK key is referenced by an existing key record, the key record is reused, otherwise a new key record is created with the TACK's key and min_generation.

2. A new name record is created containing the relevant name, an "initial timestamp" equal to the current time, and an empty "active period end".

3. If there is an existing relevant pin, the pin SHALL be deleted (see Section 5.3.6).

If there is no TACK and the relevant pin is inactive, the pin SHALL be deleted (see Section 5.3.6).

The following table summarizes this behavior based on whether the relevant pin is active and references the TACK key. The "(*)" means "if pin activation is enabled".

Pin status	Pin references TACK	Result
Active	Yes	Extend activation period (*)
Active	No (or no TACK)	Rejected
Inactive	Yes	Activate pin (*)
Inactive	No	Replace with new inactive pin
Inactive	No TACK	Delete pin
No pin	-	Create new inactive pin
No pin	No TACK	-

5.3.5. Discard pins based on break signatures

All key records broken by break signatures SHALL be discarded, along with their associated name records. A key record is broken by a break signature if the break signature passes the following checks:

1. "public_key" is referenced by the key record.

2. "signature" is a correct ECDSA signature (else "decrypt_error").

5.3.6. Deleting pins

A client might need to delete a pin from its store as a result of the algorithms in Section 5.3.4. A client MAY also delete pins from its store at any time, whether to save space, protect privacy, or for any other reason. To delete a pin, its name record SHALL be removed. If this leaves a key record with no associated name records, the key record MAY be removed as well. Pins MAY be deleted regardless of whether they are active or inactive, however for security concerns regarding pin deletion, see Section 9.2.

Deleting pins unnecessarily will reduce the benefits of TACK, so SHOULD be avoided. Note that a pin SHOULD NOT be deleted simply because it has become inactive. Instead, such a pin SHOULD be retained, so that it can be re-activated in the future by the algorithms in Section 5.3.4.

6. Variations on client processing

6.1. TACK and 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, or if the user elects to "connect anyway" despite certificate and/or pin failures.

6.2. Application-specific pinning

In addition to the hostname-based pinning described in Section 5, some applications may require "application-specific pins", where an application-layer name is pinned to a TACK key. For example, an SMTP MTA may wish to authenticate receiving MTAs by pinning email domains to the receiving MTAs' TACK keys.

Application-specific pins may require redefinition of the name record's "name" field, the "relevant name" for the TLS connection, and the "pin activation" signal. With these items redefined, the client processing rules in Section 5 may be reused.

Note that a server using application-specific pins is still subject to hostname pins, and a client MAY apply either or both forms of pinning.

The specification of application-specific pinning for particular applications is outside the scope of this document.

7. TACK IDs

A "TACK ID" MAY be used to represent a TACK public key to users in a form that is relatively easy to compare and transcribe. A TACK ID 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.

Example TACK IDs:

```
quxiz.kpldu.uuedc.j5znm.7mqst

a334f.bt7ts.ljb3b.y24ij.6zhwm

ebsx7.z22qt.okobu.ibhut.xzdny
```

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 such 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.

Signature expiration: It is convenient to set the TACK expiration equal to the end-entity certificate expiration, so that the TACK and certificate may both be replaced at the same time. Alternatively, short-lived TACKs may be used so that a compromised TLS private key has limited value to an attacker.

Break signatures: A break signature only needs to be published for a time interval equal to the maximum active period of any affected pins. For example, if a TACK has been only been published on a website for 24 hours, to remove the TACK only requires publishing the break signature for 24 hours.

Pin activation: Pin activation SHOULD only be enabled 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.

Pin deactivation: The pin_activation field can be used to phase out TACKs for a hostname. If all servers at a hostname disable pin activation, all existing pins for the hostname will eventually become inactive, at which point the servers' TACKs can be removed.

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 revocation information (min_generation updates and break signatures). Clients are advised to collaborate 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 due to an inaccurate client clock. Such methods MAY include using time synchronization protocols such as NTP [RFC5905], or accepting seemingly-expired TACKs 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 "active_period_end".

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.

[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)