Network Working Group
Internet-Draft
Intended status: Experimental
Expires: March 19, 2018

Y. Sheffer
Intuit
D. Migault
Ericsson

September 15, 2017

TLS Server Identity Pinning with Tickets
draft-sheffer-tls-pinning-ticket-05

Abstract

Misissued public-key certificates can prevent TLS clients from appropriately authenticating the TLS server. Several alternatives have been proposed to detect this situation and prevent a client from establishing a TLS session with a TLS end point authenticated with an illegitimate public-key certificate, but none is currently in wide use.

This document proposes to extend TLS with opaque pinning tickets as a way to pin the server's identity. During an initial TLS session, the server provides an original encrypted pinning ticket. In subsequent TLS session establishment, upon receipt of the pinning ticket, the server proves its ability to decrypt the pinning ticket and thus the ownership if the pinning protection key. The client can now safely conclude that the TLS session is established with the same TLS server as the original TLS session. One of the important properties of this proposal is that no manual management actions are required.

Table of Contents

1. Introduction

The weaknesses of the global PKI system are by now widely known. Essentially, any valid CA may issue a certificate for any organization without the organization's approval (a misissued or "fake" certificate), and use the certificate to impersonate the organization. There are many attempts to resolve these weaknesses, including Certificate Transparency (CT) [RFC6962], HTTP Public Key Pinning (HPKP) [RFC7469], and TACK [I-D.perrin-tls-tack]. CT requires cooperation of a large portion of the hundreds of extant certificate authorities (CAs) before it can be used "for real", in enforcing mode. It is noted that the relevant industry forum (CA/Browser Forum) is indeed pushing for such extensive adoption. TACK has some similarities to the current proposal, but work on it seems to have stalled. Section 6.2 compares our proposal to TACK.

HPKP is an IETF standard, but so far has proven hard to deploy. HPKP pins (fixes) a public key, one of the public keys listed in the certificate chain. As a result, HPKP needs to be coordinated with the certificate management process. Certificate management impacts HPKP and thus increases the probability of HPKP failures. This risk is made even higher given the fact that, even though work has been done at the ACME WG to automate certificate management, in many or even most cases, certificates are still managed manually. As a

result, HPKP cannot be completely automated resulting in error-prone manual configuration. Such errors could prevent the web server from being accessed by some clients. In addition, HPKP uses a HTTP header which makes this solution HTTPS specific and not generic to TLS. On the other hand, the current document provides a solution that is independent of the server's certificate management and that can be entirely and easily automated. Section 6.1 compares HPKP to the current draft in more detail.

The ticket pinning proposal augments these mechanisms with a much easier to implement and deploy solution for server identity pinning, by reusing some of the ideas behind TLS session resumption.

Ticket pinning is a second factor server authentication method and is not proposed as a substitute of the authentication method provided in the TLS key exchange. More specifically, the client only uses the pinning identity method after the TLS key exchange is successfully completed. In other words, the pinning identity method is only performed over an authenticated TLS session. Note that Ticket Pinning does not pin certificate information and as such should be considered a "real" independent second factor authentication.

Ticket pinning is a Trust On First Use (TOFU) mechanism, in that the first server authentication is only based on PKI certificate validation, but for any follow-on sessions, the client is further ensuring the server's identity based on the server's ability to decrypt the ticket, in addition to normal PKI certificate authentication.

During initial TLS session establishment, the client requests a pinning ticket from the server. Upon receiving the request the server generates a pinning secret which is expected to be unpredictable for peers other than the client or the server. In our case, the pinning secret is generated from parameters exchanged during the TLS key exchange, so client and server can generate it locally and independently. The server constructs the pinning ticket with the necessary information to retrieve the pinning secret. The server then encrypts the ticket and returns the pinning ticket to the client with an associated pinning lifetime.

The pinning lifetime value indicates for how long the server promises to retain the server-side ticket-encryption key, which allows it to complete the protocol exchange correctly and prove its identity. The committed lifetime is typically on the order of weeks or months.

Once the key exchange is completed and the server is deemed authenticated, the client generates locally the pinning secret and

caches the server's identifiers to index the pinning secret as well as the pinning ticket and its associated lifetime.

When the client re-establishes a new TLS session with the server, it sends the pinning ticket to the server. Upon receiving it, the server returns a proof of knowledge of the pinning secret. Once the key exchange is completed and the server has been authenticated, the client checks the pinning proof returned by the server using the client's stored pinning secret. If the proof matches, the client can conclude that the server it is currently connecting to is in fact the correct server.

This version of the draft only applies to TLS 1.3. We believe that the idea can also be back-fitted into earlier versions of the protocol.

The main advantages of this protocol over earlier pinning solutions are:

- The protocol is at the TLS level, and as a result is not restricted to HTTP at the application level.

- The protocol is robust to server IP, CA, and public key changes. The server is characterized by the ownership of the pinning protection key, which is never provided to the client. Server configuration parameters such as the CA and the public key may change without affecting the pinning ticket protocol.

- Once a single parameter is configured (the ticket's lifetime), operation is fully automated. The server administrator need not bother with the management of backup certificates or explicit pins.

- For server clusters, we reuse the existing [RFC5077] infrastructure where it exists.

- Pinning errors, presumably resulting from MITM attacks, can be detected both by the client and the server. This allows for server-side detection of MITM attacks using large-scale analytics, and with no need to rely on clients to explicitly report the error.

A note on terminology: unlike other solutions in this space, we do not do "certificate pinning" (or "public key pinning"), since the protocol is oblivious to the server's certificate. We prefer the term "server identity pinning" for this new solution. In out solution, the server proves its identity by generating a proof that it can read and decrypt an encrypted ticket. As a result, the

identity proof relies on proof of ownership of the pinning protection key. However, this key is never exchanged with the client or known by it, and so cannot itself be pinned.

1.1. Conventions used in this document

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].

2. Protocol Overview

The protocol consists of two phases: the first time a particular client connects to a server, and subsequent connections.

This protocol supports full TLS handshakes, as well as 0-RTT handshakes. Below we present it in the context of a full handshake, but behavior in 0-RTT handshakes should be identical.

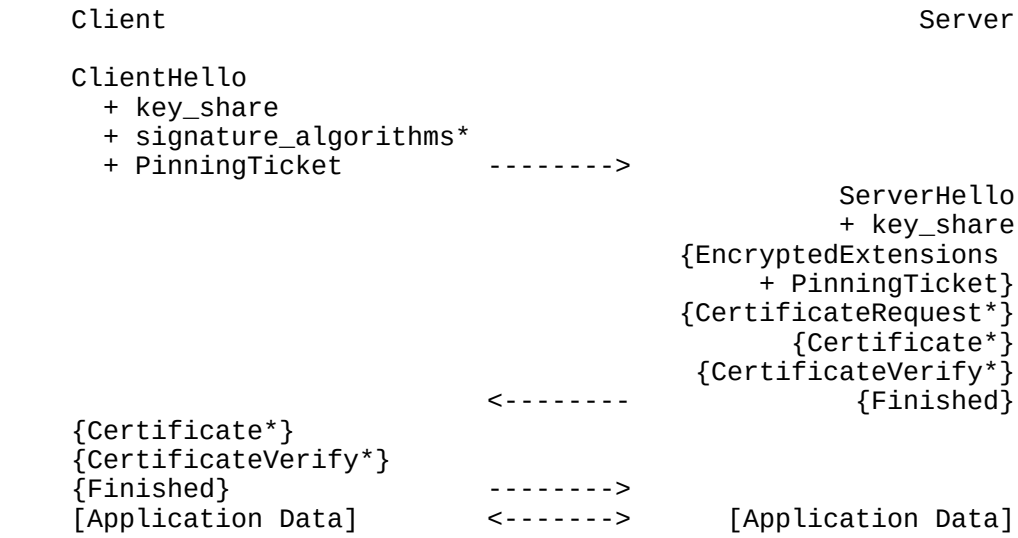
The document presents some similarities with the ticket resumption mechanism described in [RFC5077]. However the scope of this document differs from session resumption mechanisms implemented with [RFC5077] or with other mechanisms. Specifically, the pinning ticket does not carry any state associated with a TLS session and thus cannot be used for session resumption, or to authenticate the client. Instead, the pinning ticket only contains the Pinning Secret used to generate the proof.

With TLS 1.3, session resumption is based on a preshared key (PSK). This is orthogonal to this protocol. With TLS 1.3, a TLS session can be established using PKI and a pinning ticket, and later resumed with PSK.

However, the protocol described in this document addresses the problem of misissued certificates. Thus, it is not expected to be used outside a certificate-based TLS key exchange, such as in PSK. As a result, PSK handshakes MUST NOT include the extension defined here.

2.1. Initial Connection

When a client first connects to a server, it requests a pinning ticket by sending an empty PinningTicket extension, and receives it as part of the server’s first response, in the returned PinningTicket extension.



* Indicates optional or situation-dependent messages that are not always sent.

{} Indicates messages protected using keys derived from the ephemeral secret.

[] Indicates messages protected using keys derived from the master secret.

If a client supports the pinning ticket extension and does not have any pinning ticket associated with the server, the exchange is considered as an initial connection. Other reasons the client may not have a pinning ticket include the client having flushed its pinning ticket store, or the committed lifetime of the pinning ticket having expired.

Upon receipt of the PinningTicket extension, the server computes a pinning secret (Section 4.1), and sends the pinning ticket (Section 4.2) encrypted with the pinning protection key (Section 4.3). The pinning ticket is associated with a lifetime value by which the server assumes the responsibility of retaining the pinning protection key and being able to decrypt incoming pinning tickets during the period indicated by the committed lifetime.

Once the pinning ticket has been generated, the server returns the pinning ticket and the committed lifetime in a PinningTicket extension embedded in the EncryptedExtensions message. We note that a PinningTicket extension MUST NOT be sent as part of a HelloRetryRequest.

Upon receiving the pinning ticket, the client MUST NOT accept it until the key exchange is completed and the server authenticated. If the key exchange is not completed successfully, the client MUST ignore the received pinning ticket. Otherwise, the client computes the pinning secret and SHOULD cache the pinning secret and the pinning ticket for the duration indicated by the pinning ticket lifetime. The client SHOULD clean up the cached values at the end of the indicated lifetime.

2.2. Subsequent Connections

When the client initiates a connection to a server it has previously seen (see Section 2.3 on identifying servers), it SHOULD send the pinning ticket for that server. The pinning ticket, pinning secret and pinning ticket lifetime computed during the establishment of the previous TLS session are designated in this document as the "original" ones, to distinguish them from a new ticket that may be generated during the current session.

The server MUST extract the original pinning_secret value from the ticket and MUST respond with a PinningTicket extension, which includes:

- A proof that the server can understand the ticket that was sent by the client; this proof also binds the pinning ticket to the server’s (current) public key, as well as the ongoing TLS session. The proof is MANDATORY if a pinning ticket was sent by the client.

- A fresh pinning ticket. The main reason for refreshing the ticket on each connection is privacy: to avoid the ticket serving as a fixed client identifier. It is RECOMMENDED to include a fresh ticket with each response.

If the server cannot validate the received ticket, that might indicate an earlier MITM attack on this client. The server MUST then abort the connection with a handshake_failure alert, and SHOULD log this failure.

The client MUST verify the proof, and if it fails to do so, MUST issue a handshake_failure alert and abort the connection (see also Section 8.5). It is important that the client does not attempt to "fall back" by omitting the PinningTicket extension.

When the connection is successfully set up, i.e. after the Finished message is verified, the client SHOULD store the new ticket along with the corresponding pinning_secret, replacing the original ticket.

Although this is an extension, if the client already has a ticket for a server, the client MUST interpret a missing PinningTicket extension in the server's response as an attack, because of the server's prior commitment to respect the ticket. The client MUST abort the connection in this case. See also Section 5.5 on ramping down support for this extension.

2.3. Indexing the Pins

Each pin is associated with a host name, protocol (TLS or DTLS) and port number. In other words, the pin for port TCP/443 may be different from that for DTLS or from the pin for port TCP/8443. The host name MUST be the value sent inside the Server Name Indication (SNI) extension. This definition is similar to a Web Origin [RFC6454], but does not assume the existence of a URL.

The purpose of ticket pinning is to pin the server identity. As a result, any information orthogonal to the server's identity MUST NOT be considered in indexing. More particularly, IP addresses are ephemeral and forbidden in SNI and therefore pins MUST NOT be associated with IP addresses. Similarly, CA names or public keys associated with server MUST NOT be used for indexing as they may change over time.

3. Message Definitions

This section defines the format of the PinningTicket extension. We follow the message notation of [I-D.ietf-tls-tls13].

```
 opaque pinning_ticket<0..2^16-1>;

 opaque pinning_proof<0..2^8-1>;

 struct {
   select (Role) {
   case client:
       pinning_ticket ticket<0..2^16-1>; //omitted on 1st connection

     case server:
       pinning_proof proof<0..2^8-1>; //no proof on 1st connection
       pinning_ticket ticket<0..2^16-1>; //omitted on ramp down
       uint32 lifetime;
   }
} PinningTicketExtension;
```

ticket: a pinning ticket sent by the client or returned by the server. The ticket is opaque to the client. The extension MUST contain exactly 0 or 1 tickets.

proof: a demonstration by the server that it understands the received ticket and therefore that it is in possession of the secret that was used to generate it originally. The extension MUST contain exactly 0 or 1 proofs.

lifetime: the duration (in seconds) that the server commits to accept offered tickets in the future.

4. Cryptographic Operations

This section provides details on the cryptographic operations performed by the protocol peers.

4.1. Pinning Secret

The pinning secret is generated locally by the client and the server which means they must use the same inputs to generate it. This value must be generated before the ServerHello message is sent, as the server includes the corresponding pinning ticket in the ServerHello message. In addition, the pinning secret must be unpredictable to any party other than the client and the server.

The pinning secret is derived using the Derive-Secret function provided by TLS 1.3, described in Section "Key Schedule" of [I-D.ietf-tls-tls13].

pinning secret = Derive-Secret(Handshake Secret, "pinning secret", ClientHello...ServerHello)

4.2. Pinning Ticket

The pinning ticket contains the pinning secret. The pinning ticket is provided by the client to the server which decrypts it in order to extract the pinning secret and responds with a pinning proof. As a result, the characteristics of the pinning ticket are:

- Pinning tickets MUST be encrypted and integrity-protected using strong cryptographic algorithms.

- Pinning tickets MUST be protected with a long-term pinning protection key.

- Pinning tickets MUST include a pinning protection key ID or serial number as to enable the pinning protection key to be refreshed.

- The pinning ticket MAY include other information, in addition to the pinning secret.

The pinning ticket's format is not specified by this document, but we RECOMMEND a format similar to the one proposed by [RFC5077].

4.3. Pinning Protection Key

The pinning protection key is only used by the server and so remains server implementation specific. [RFC5077] recommends the use of two keys, but when using AEAD algorithms only a single key is required.

When a single server terminates TLS for multiple virtual servers using the Server Name Indication (SNI) mechanism, we strongly RECOMMEND to use a separate protection key for each one of them, in order to allow migrating virtual servers between different servers while keeping pinning active.

As noted in Section 5.1, if the server is actually a cluster of machines, the protection key MUST be synchronized between all the nodes that accept TLS connections to the same server name. When [RFC5077] is deployed, an easy way to do it is to derive the protection key from the session-ticket protection key, which is already synchronized. For example:

```
pinning_protection_key = HKDF-Expand(resumption_protection_key,
                                     "pinning protection", L)
```

4.4. Pinning Proof

The pinning proof is sent by the server to demonstrate that it has been able to decrypt the pinning ticket and retrieve the pinning secret. The proof must be unpredictable and must not be replayed. Similarly to the pinning secret, the pinning proof is sent by the server in the ServerHello message. In addition, it must not be possible for a MITM server with a fake certificate to obtain a pinning proof from the original server.

In order to address these requirements, the pinning proof is bound to the TLS session as well as the public key of the server:

```
proof = HMAC(original_pinning_secret, "pinning proof" +
             Handshake-Secret + Hash(server_public_key))
```

where HMAC [RFC2104] uses the Hash algorithm that was negotiated in the handshake, and the same hash is also used over the server's public key. The original_pinning_secret value refers to the secret value extracted from the ticket sent by the client, to distinguish it from a new pinning secret value that is possibly computed in the current exchange. The server_public_key value is the DER

representation of the public key, specifically the SubjectPublicKeyInfo structure as-is.

5. Operational Considerations

The main motivation behind the current protocol is to enable identity pinning without the need for manual operations. To achieve this goal operations described in identity pinning are only performed within the current TLS session, and there is no dependence on any TLS configuration parameters such as CA identity or public keys. As a result, configuration changes are unlikely to lead to desynchronized state between the client and the server. Manual operations are susceptible to human error and in the case of public key pinning, can easily result in "server bricking": the server becoming inaccessible to some or all of its users.

5.1. Protection Key Synchronization

The only operational requirement when deploying this protocol is that if the server is part of a cluster, protection keys (the keys used to encrypt tickets) MUST be synchronized between all cluster members. The protocol is designed so that if resumption ticket protection keys [RFC5077] are already synchronized between cluster members, nothing more needs to be done.

Moreover, synchronization does not need to be instantaneous, e.g. protection keys can be distributed a few minutes or hours in advance of their rollover. In such scenarios, each cluster member MUST be able to accept tickets protected with a new version of the protection key, even while it is still using an old version to generate keys. This ensures that a client that receives a "new" ticket does not next hit a cluster member that still rejects this ticket.

Misconfiguration can lead to the server’s clock being off by a large amount of time. Therefore we RECOMMEND never to automatically delete protection keys, even when they are long expired.

5.2. Ticket Lifetime

The lifetime of the ticket is a commitment by the server to retain the ticket’s corresponding protection key for this duration, so that the server can prove to the client that it knows the secret embedded in the ticket. For production systems, the lifetime SHOULD be between 7 and 31 days.

5.3. Certificate Renewal

The protocol ensures that the client will continue speaking to the correct server even when the server's certificate is renewed. In this sense, we are not "pinning certificates" and the protocol should more precisely be called "server identity pinning".

Note that this property is not impacted by the use of the server's public key in the pinning proof, because the scope of the public key used is only the current TLS session.

5.4. Certificate Revocation

The protocol is orthogonal to certificate validation in the sense that, if the server's certificate has been revoked or is invalid for some other reason, the client MUST refuse to connect to it regardless of any ticket-related behavior.

5.5. Disabling Pinning

A server implementing this protocol MUST have a "ramp down" mode of operation where:

- The server continues to accept valid pinning tickets and responds correctly with a proof.

- The server does not send back a new pinning ticket.

After a while no clients will hold valid tickets any more and the feature may be disabled. Note that clients that do not receive a new pinning ticket do not remove the original ticket. Instead, the client keeps on using the ticket until its lifetime expires.

Issuing a new pinning ticket with a shorter lifetime would only delay the ramp down process, as the shorter lifetime can only affect clients that actually initiated a new connection. Other clients would still see the original lifetime for their pinning tickets.

5.6. Server Compromise

If a server compromise is detected, the pinning protection key MUST be rotated immediately, but the server MUST still accept valid tickets that use the old, compromised key. Clients that still hold old pinning tickets will remain vulnerable to MITM attacks, but those that connect to the correct server will immediately receive new tickets protected with the newly generated pinning protection key.

The same procedure applies if the pinning protection key is compromised directly, e.g. if a backup copy is inadvertently made public.

5.7. Disaster Recovery

All web servers in production need to be backed up, so that they can be recovered if a disaster (including a malicious activity) ever wipes them out. Backup typically includes the certificate and its private key, which must be backed up securely. The pinning secret, including earlier versions that are still being accepted, must be backed up regularly. However since it is only used as an authentication second factor, it does not require the same level of confidentiality as the server's private key.

Readers should note that [RFC5077] session resumption keys are more security sensitive, and should normally not be backed up but rather treated as ephemeral keys. Even when servers derive pinning secrets from resumption keys (Section 4.1), they MUST NOT back up resumption keys.

6. Previous Work

This section compares ticket pinning to two earlier proposals, HPKP and TACK.

6.1. Comparison: HPKP

The current IETF standard for pinning the identity of web servers is the Public Key Pinning Extension for HTTP, or HPKP [RFC7469].

The main differences between HPKP and the current document are the following:

- HPKP limits its scope to HTTPS, while the current document considers all application above TLS.

- HPKP pins the public key of the server (or another public key along the certificate chain) and as such is highly dependent on the management of certificates. Such dependency increases the potential error surface, especially as certificate management is not yet largely automated. The current proposal, on the other hand is independent of certificate management.

- HPKP pins public keys which are public and used for the standard TLS authentication. Identity pinning relies on the ownership of the pinning key which is not disclosed to the public and not involved in the standard TLS authentication. As a result,

identity pinning is a completely independent second factor authentication mechanism.

- HPKP relies on a backup key to recover the mis-issuance of a key. We believe such backup mechanisms add excessive complexity and cost. Reliability of the current mechanism is primarily based on its being highly automated.

- HPKP relies on the client to report errors to the report-uri. The current document not need any out-of band mechanism, and the server is informed automatically. This provides an easier and more reliable health monitoring.

On the other hand, HPKP shares the following aspects with identity pinning:

- Both mechanisms provide hard failure. With HPKP only the client is aware of the failure, while with the current proposal both client and server are informed of the failure. This provides room for further mechanisms to automatically recover such failures.

- Both mechanisms are subject to a server compromise in which users are provided with an invalid ticket (e.g. a random one) or HTTP Header, with a very long lifetime. For identity pinning, this lifetime cannot be longer than 31 days. In both cases, clients will not be able to reconnect the server during this lifetime. With the current proposal, an attacker needs to compromise the TLS layer, while with HPKP, the attacker needs to compromise the HTTP server. Arguably, the TLS-level compromise is typically more difficult for the attacker.

Unfortunately HPKP has not seen wide deployment yet. As of March 2016, the number of servers using HPKP was less than 3000 [Netcraft]. This may simply be due to inertia, but we believe the main reason is the interactions between HPKP and manual certificate management which is needed to implement HPKP for enterprise servers. The penalty for making mistakes (e.g. being too early or too late to deploy new pins) is having the server become unusable for some of the clients.

To demonstrate this point, we present a list of the steps involved in deploying HPKP on a security-sensitive Web server.

1. Generate two public/private key-pairs on a computer that is not the Live server. The second one is the "backup1" key-pair.

   "openssl genrsa -out "example.com.key" 2048;"

   "openssl genrsa -out "example.com.backup1.key" 2048;"

2. Generate hashes for both of the public keys. These will be used in the HPKP header:

   "openssl rsa -in "example.com.key" -outform der -pubout | openssl dgst -sha256 -binary | openssl enc -base64"

   "openssl rsa -in "example.com.backup1.key" -outform der -pubout | openssl dgst -sha256 -binary | openssl enc -base64"

3. Generate a single CSR (Certificate Signing Request) for the first key-pair, where you include the domain name in the CN (Common Name) field:

   "openssl req -new -subj "/C=GB/ST=Area/L=Town/O=Company/ CN=example.com" -key "example.com.key" -out "example.com.csr";"

4. Send this CSR to the CA (Certificate Authority), and go though the dance to prove you own the domain. The CA will give you back a single certificate that will typically expire within a year or two.

5. On the Live server, upload and setup the first key-pair (and its certificate). At this point you can add the "Public-Key-Pins" header, using the two hashes you created in step 2.

   Note that only the first key-pair has been uploaded to the server so far.

6. Store the second (backup1) key-pair somewhere safe, probably somewhere encrypted like a password manager. It won’t expire, as it’s just a key-pair, it just needs to be ready for when you need to get your next certificate.

7. Time passes... probably just under a year (if waiting for a certificate to expire), or maybe sooner if you find that your server has been compromised and you need to replace the key-pair and certificate.

8. Create a new CSR (Certificate Signing Request) using the "backup1" key-pair, and get a new certificate from your CA.

9. Generate a new backup key-pair (backup2), get its hash, and store it in a safe place (again, not on the Live server).

10. Replace your old certificate and old key-pair, and update the "Public-Key-Pins" header to remove the old hash, and add the new "backup2" key-pair.

Note that in the above steps, both the certificate issuance as well as the storage of the backup key pair involve manual steps. Even with an automated CA that runs the ACME protocol, key backup would be a challenge to automate.

6.2. Comparison: TACK

Compared with HPKP, TACK [I-D.perrin-tls-tack] is a lot more similar to the current draft. It can even be argued that this document is a symmetric-cryptography variant of TACK. That said, there are still a few significant differences:

- Probably the most important difference is that with TACK, validation of the server certificate is no longer required, and in fact TACK specifies it as a "MAY" requirement (Sec. 5.3). With ticket pinning, certificate validation by the client remains a MUST requirement, and the ticket acts only as a second factor. If the pinning secret is compromised, the server's security is not immediately at risk.

- Both TACK and the current draft are mostly orthogonal to the server certificate as far as their life cycle, and so both can be deployed with no manual steps.

- TACK uses ECDSA to sign the server's public key. This allows cooperating clients to share server assertions between themselves. This is an optional TACK feature, and one that cannot be done with pinning tickets.

- TACK allows multiple servers to share its public keys. Such sharing is disallowed by the current document.

- TACK does not allow the server to track a particular client, and so has better privacy properties than the current draft.

- TACK has an interesting way to determine the pin's lifetime, setting it to the time period since the pin was first observed, with a hard upper bound of 30 days. The current draft makes the lifetime explicit, which may be more flexible to deploy. For example, Web sites which are only visited rarely by users may opt for a longer period than other sites that expect users to visit on a daily basis.

7. Implementation Status

Note to RFC Editor: please remove this section before publication, including the reference to [RFC7942].

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC7942]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 7942, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

7.1. Mint Fork

7.1.1. Overview

A fork of the Mint TLS 1.3 implementation, developed by Yaron Sheffer and available at https://github.com/yaronf/mint.

7.1.2. Description

This is a fork of the TLS 1.3 implementation, and includes client and server code. In addition to the actual protocol, several utilities are provided allowing to manage pinning protection keys on the server side, and pinning tickets on the client side.

7.1.3. Level of Maturity

This is a prototype.

7.1.4. Coverage

The entire protocol is implemented.

7.1.5. Version Compatibility

The implementation is compatible with draft-sheffer-tls-pinning-ticket-02.

7.1.6. Licensing

Mint itself and this fork are available under an MIT license.

7.1.7. Contact Information

See author details below.

8. Security Considerations

This section reviews several security aspects related to the proposed extension.

8.1. Trust on First Use (TOFU) and MITM Attacks

This protocol is a "trust on first use" protocol. If a client initially connects to the "right" server, it will be protected against MITM attackers for the lifetime of each received ticket. If it connects regularly (depending of course on the server-selected lifetime), it will stay constantly protected against fake certificates.

However if it initially connects to an attacker, subsequent connections to the "right" server will fail. Server operators might want to advise clients on how to remove corrupted pins, once such large scale attacks are detected and remediated.

The protocol is designed so that it is not vulnerable to an active MITM attacker who has real-time access to the original server. The pinning proof includes a hash of the server’s public key, to ensure the client that the proof was in fact generated by the server with which it is initiating the connection.

8.2. Pervasive Monitoring

Some organizations, and even some countries perform pervasive monitoring on their constituents [RFC7258]. This often takes the form of always-active SSL proxies. Because of the TOFU property, this protocol does not provide any security in such cases.

8.3. Server-Side Error Detection

Uniquely, this protocol allows the server to detect clients that present incorrect tickets and therefore can be assumed to be victims of a MITM attack. Server operators can use such cases as indications of ongoing attacks, similarly to fake certificate attacks that took place in a few countries in the past.

8.4. Client Policy and SSL Proxies

Like it or not, some clients are normally deployed behind an SSL proxy. Similarly to [RFC7469], it is acceptable to allow pinning to be disabled for some hosts according to local policy. For example, a UA MAY disable pinning for hosts whose validated certificate chain terminates at a user-defined trust anchor, rather than a trust anchor built-in to the UA (or underlying platform). Moreover, a client MAY accept an empty PinningTicket extension from such hosts as a valid response.

8.5. Client-Side Error Behavior

When a client receives a malformed or empty PinningTicket extension from a pinned server, it MUST abort the handshake and MUST NOT retry with no PinningTicket in the request. Doing otherwise would expose the client to trivial fallback attacks, similar to those described in [RFC7507].

This rule can however have negative affects on clients that move from behind SSL proxies into the open Internet and vice versa, if the advice in Section 8.4 is not followed. Therefore, we RECOMMEND that browser and library vendors provide a documented way to remove stored pins.

8.6. Stolen and Forged Tickets

Stealing pinning tickets even in conjunction with other pinning parameters, such as the associated pinning secret, provides no benefit to the attacker since pinning tickets are used to secure the client rather than the server. Similarly, it is useless to forge a ticket for a particular sever.

8.7. Client Privacy

This protocol is designed so that an external attacker cannot correlate between different requests of a single client, provided the client requests and receives a fresh ticket upon each connection.

On the other hand, the server to which the client is connecting can easily track the client. This may be an issue when the client expects to connect to the server (e.g., a mail server) with multiple identities. Implementations SHOULD allow the user to opt out of pinning, either in general or for particular servers.

8.8. Ticket Protection Key Management

While the ticket format is not mandated by this document, we RECOMMEND using authenticated encryption to protect it. Some of the algorithms commonly used for authenticated encryption, e.g. GCM, are highly vulnerable to nonce reuse, and this problem is magnified in a cluster setting. Therefore implementations that choose AES-128-GCM MUST adopt one of these two alternatives:

- Partition the nonce namespace between cluster members and use monotonic counters on each member, e.g. by setting the nonce to the concatenation of the cluster member ID and an incremental counter.

- Generate random nonces but avoid the so-called birthday bound, i.e. never generate more than 2**64 encrypted tickets for the same ticket pinning protection Key.

An alternative design which has been attributed to Karthik Bhargavan is as follows. Start with a 128-bit master key "K_master" and then for each encryption, generate a 256-bit random nonce and compute:

```
K = HKDF(K_master, Nonce || "key")
N = HKDF(K_master, Nonce || "nonce")
```

And use these values to encrypt the ticket, AES-GCM(K, N, <data>).

9. IANA Considerations

IANA is requested to allocate a TicketPinning extension value in the TLS ExtensionType Registry.

No registries are defined by this document.

10. Acknowledgements

The original idea behind this proposal was published in [Oreo] by Moty Yung, Benny Pinkas and Omer Berkman. The current protocol is but a distant relative of the original Oreo protocol, and any errors are the draft authors' alone.

We would like to thank Dave Garrett, Daniel Kahn Gillmor, Eric Rescorla and Yoav Nir for their comments on this draft. Special thanks to Craig Francis for contributing the HPKP deployment script, and to Ralph Holz for several fruitful discussions.

11. References

11.1. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-21 (work in progress), July 2016.

[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>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <http://www.rfc-editor.org/info/rfc5077>.

11.2. Informative References

[I-D.perrin-tls-tack]
Marlinspike, M., "Trust Assertions for Certificate Keys", draft-perrin-tls-tack-02 (work in progress), January 2013.

[Netcraft]
Mutton, P., "HTTP Public Key Pinning: You’re doing it wrong!", March 2016, <http://news.netcraft.com/archives/2016/03/30/http-public-key-pinning-youre-doing-it-wrong.html>.

[Oreo] Berkman, O., Pinkas, B., and M. Yung, "Firm Grip Handshakes: A Tool for Bidirectional Vouching", Cryptology and Network Security, pp. 142-157 , 2012.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <http://www.rfc-editor.org/info/rfc6454>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <http://www.rfc-editor.org/info/rfc6962>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RFC7469] Evans, C., Palmer, C., and R. Sleevi, "Public Key Pinning Extension for HTTP", RFC 7469, DOI 10.17487/RFC7469, April 2015, <http://www.rfc-editor.org/info/rfc7469>.

[RFC7507] Moeller, B. and A. Langley, "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks", RFC 7507, DOI 10.17487/RFC7507, April 2015, <http://www.rfc-editor.org/info/rfc7507>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <http://www.rfc-editor.org/info/rfc7942>.

Appendix A. Document History

A.1. draft-sheffer-tls-pinning-ticket-05

- Multiple comments from Eric Rescorla.

A.2. draft-sheffer-tls-pinning-ticket-04

- Editorial changes.
- Two-phase rotation of protection key.

A.3. draft-sheffer-tls-pinning-ticket-03

- Deleted redundant length fields in the extension's formal definition.
- Modified cryptographic operations to align with the current state of TLS 1.3.
- Numerous textual improvements.

A.4. draft-sheffer-tls-pinning-ticket-02

- Added an Implementation Status section.
- Added lengths into the extension structure.
- Changed the computation of the pinning proof to be more robust.
- Clarified requirements on the length of the pinning_secret.
- Revamped the HPKP section to be more in line with current practices, and added recent statistics on HPKP deployment.

A.5. draft-sheffer-tls-pinning-ticket-01

- Corrected the notation for variable-sized vectors.
- Added a section on disaster recovery and backup.
- Added a section on privacy.
- Clarified the assumptions behind the HPKP procedure in the comparison section.
- Added a definition of pin indexing (origin).

- Adjusted to the latest TLS 1.3 notation.

A.6. draft-sheffer-tls-pinning-ticket-00

Initial version.

Authors' Addresses

Yaron Sheffer
Intuit

EMail: yaronf.ietf@gmail.com

Daniel Migault
Ericsson

EMail: daniel.migault@ericsson.com