Token Binding for 0-RTT TLS 1.3 Connections
draft-ietf-tokbind-tls13-0rtt-00

Abstract

   This document describes how Token Binding can be used in the 0-RTT
   data of a TLS 1.3 connection. This involves updating how Token
   Binding negotiation works and adding a mechanism for indicating
   whether a server prevents replay. A TokenBindingMessage sent in
   0-RTT data has different security properties than one sent after the
   TLS handshake has finished, which this document also describes.

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 May 21, 2017.

Table of Contents

1. Introduction

Token Binding ([I-D.ietf-tokbind-protocol]) cryptographically binds
security tokens (e.g. HTTP cookies, OAuth tokens) to the TLS layer
on which they are presented. It does so by signing an [RFC5705]
exporter value from the TLS connection. TLS 1.3 introduces a new
mode that allows a client to send application data on its first
flight. If this 0-RTT data contains a security token, then a client
using Token Binding would want to prove possession of its Token
Binding private key so that the server can verify the binding. The
[RFC5705]-style exporter provided by TLS 1.3 cannot be run until the
handshake has finished. TLS 1.3 also provides an exporter that can
be used with 0-RTT data, but it requires that the application
explicitly specify that use. This document specifies how to use the
early_exporter_secret with Token Binding in TLS 1.3 0-RTT data.

1.1. Requirements Language

   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. Proposed Design

   A TokenBinding struct as defined in [I-D.ietf-tokbind-protocol]
   contains a signature of the EKM value from the TLS layer. Under
   normal circumstances, a TokenBinding on a TLS 1.3 connection would
   use the exporter_secret to derive the EKM value. When 0-RTT data is
   assembled to be sent, the exporter_secret is not yet available. This
   design changes the definition of the TokenBinding.signature field to
   use the exporter with early_exporter_secret for 0-RTT data. Since no
   negotiation for the connection can happen before the client sends
   this TokenBindingMessage in 0-RTT data, this document also describes
   how a client decides what TokenBindingMessage to send in 0-RTT data
   and how a server should interpret that message.

   If a client does not send any 0-RTT data, or if the server rejects
   the client's 0-RTT data, then the client MUST use the 1-RTT exporter,
   as defined in [I-D.ietf-tokbind-protocol].

2.1. TokenBinding Signature Definition

   In [I-D.ietf-tokbind-protocol], the signature field of the
   TokenBinding struct is defined to be the signature of a
   concatentation that includes the EKM value. Depending on the
   circumstances, the exporter value in section 7.3.3 of
   [I-D.ietf-tls-tls13] is computed using either exporter_secret or
   early_exporter_secret as the Secret. The same Secret is used for the
   entirety of the connection.

   The rules for a client choosing which exporter to use are as follows.
   A client which is not sending any 0-RTT data on a connection MUST use
   the exporter defined in [I-D.ietf-tls-tls13] (using exporter_secret
   as the Secret) for all TokenBindingMessages on that connection so
   that it is compatible with [I-D.ietf-tokbind-protocol]. A client
   that sends a TokenBindingMessage in 0-RTT data must use the exporter
   with early_exporter_secret as the Secret (the "0-RTT exporter") since
   exporter_secret is not defined at that time. A client that sends
   0-RTT data which is not rejected by the server MUST use the 0-RTT
   exporter for the rest of the connection. If the server rejects the
   client's 0-RTT data, then the client MUST use the exporter defined in
   [I-D.ietf-tls-tls13] (using exporter_secret as the Secret) for the
   remainder of the connection, as if no 0-RTT data had ever been sent.

2.2. Negotiating Token Binding

2.2.1. Negotiation TLS Extension

   The behavior of the Token Binding negotiation TLS extension does not
   change for a 0-RTT connection: the client and server should process
   this extension the same way regardless of whether the client also
   sent the EarlyDataIndication extension.

   For the sake of choosing a key parameter to use in 0-RTT data, the
   client MUST use the same key parameter that was used on the
   connection during which the ticket (now being used for resumption)
   was established. The server MUST NOT accept early data if the
   negotiated Token Binding key parameter does not match the parameter
   from the initial connection. This is the same behavior as ALPN and
   SNI extensions.

   If 0-RTT data is being sent with Token Binding using a PSK obtained
   out-of-band, then the Token Binding key parameter to use with that
   PSK must also be provisioned to both parties, and only that key
   parameter must be used with that PSK.

2.2.2. Replay Protection Indication Extension

   The signed exporter value used in a 0-RTT connection is not
   guaranteed to be unique to the connection, so an attacker may be able
   to replay the signature without having possession of the private key.
   To combat this attack, a server may implement some sort of replay
   prevention, and indicate this to the client. A new TLS extension
   "token_binding_replay_indication" is defined for the client to query
   and server to indicate whether it has implemented a mechanism to
   prevent replay.

   enum {
       token_binding_replay_indication(TBD), (65535)
   } ExtensionType;

   When sent, this extension always has zero length. If a client wishes
   to know whether its peer is preventing replay of TokenBinding structs
   across multiple connections, the client can include this extension in
   its ClientHello. Upon receiving this extension, the server must echo
   it back if it is using such a mechanism (like those described in
   Section 6.3.1) to prevent replay. A client that only wishes to send
   0-RTT Token Binding if the server implements replay protection can
   send this extension on first connection establishment, and if the
   server doesn't send it back (but does support Token Binding) the
   client can choose to not send 0-RTT messages to that server.

A client that wishes to use this extension should send it every time
it sends a "token_binding" [I-D.ietf-tokbind-negotiation] extension.

3. Implementation Challenges

The client has to be able to modify the message it sends in 0-RTT
data if the 0-RTT data gets rejected and needs to be retransmitted in
1-RTT data. Even if the Token Binding integration with 0-RTT were
modified so that Token Binding never caused a 0-RTT reject that
required rewriting a request, the client still has to handle the
server rejecting the 0-RTT data for other reasons.

HTTP2 allows for requests to different domains to share the same TLS
connection if the SAN of the cert covers those domains. If
one.example.com supports 0-RTT and Token Binding, but two.example.com
only supports Token Binding as defined in
[I-D.ietf-tokbind-protocol], those servers cannot share a cert and
use HTTP2.

4. Alternatives Considered

4.1. Use Both 0-RTT and 1-RTT Exporters on Same Connection

The client could be required to use the 0-RTT EKM when the
TokenBindingMessage is sent in 0-RTT data, and the 1-RTT EKM when it
is sent in 1-RTT data. This requires that the abstraction of the TLS
layer visible to the application where it is handling Token Binding
exposes which phase the application data is being sent/received in.
An application could very easily have this detail abstracted away;
for example, the client might have a function like
"write_possibly_early" that will send data in 0-RTT the current
connection state permits it, and otherwise send data post-handshake.
A pathological client might send the first few bytes of an
application message in 0-RTT, but send the rest after the handshake
(including the TokenBindingMessage). The server's application layer
would have to track which bytes of the request were sent pre- and
post-handshake to know how to validate that TokenBindingMessage.

This constraint could be relaxed slightly. A ratcheting mechanism
could be used where the client uses the 0-RTT EKM while it thinks
that it's writing early data (even if it isn't writing early data),
and once it knows the handshake is finished, it uses the 1-RTT EKM.
Once the server sees a TokenBindingMessage using the 1-RTT EKM, the
server would no longer accept the 0-RTT EKM. In practice, this is
difficult to implement because multiple HTTP/2 streams can be
multiplexed on the same connection, requiring the ratchet to be
synchronized across the streams.

Relaxing this further where the server will always accept either the
0-RTT or 1-RTT EKM (but the client keeps the behavior as above) is
another possibility. This is more complicated than always using the
0-RTT exporter, and provides no additional security benefits (since
the server would have to accept a client only using the 0-RTT
exporter).

4.2. Don't Remember Key Parameter From Previous Connection

The proposed design uses the same Token Binding key parameter from
the previous connection, and the TLS extension must negotiate the
same key parameter as the previous connection. This mirrors how ALPN
is negotiated in TLS 1.3. Instead of remembering this parameter, the
client could put the in first entry of their key parameters list the
key type being used in 0-RTT, and allow the client and server to
potentially negotiate a new type to use once the handshake is
complete. This alternate gains a slight amount of key type agility
in exchange for implementation difficulty. Other variations of this
are also possible, for example requiring the server to reject early
data if it doesn't choose the first parameter, or requiring the
client to send only one key parameter.

4.3. Token Binding and 0-RTT Data Are Mutually Exclusive

If a TokenBindingMessage is never allowed in 0-RTT data, then no
changes are needed to the exporter or negotiation. A server that
wishes to support Token Binding must not create any NewSessionTicket
messages with the allow_early_data flag set. A client must not send
the token binding negotiation extension and the EarlyDataIndication
extension in the same ClientHello.

5. IANA Considerations

This document defines a new TLS extension
"token_binding_replay_indication", which needs to be added to the
IANA "Transport Layer Security (TLS) Extensions" registry.

6. Security Considerations

Token Binding messages that use the 0-RTT exporter have weaker
security properties than with the [RFC5705] exporter. If either
party of a connection using Token Binding does not wish to use 0-RTT
token bindings, they can do so: a client can choose to never send
0-RTT data on a connection where it uses token binding, and a server
can choose to reject any 0-RTT data sent on a connection that
negotiated token binding.

0-RTT data in TLS 1.3 has weaker security properties than other kinds
of TLS data. Specifically, TLS 1.3 does not guarantee non-
replayability of data between connections. Token Binding has similar
replayability issues when in 0-RTT data, but preventing replay of
Token Binding and preventing replay of 0-RTT data are two separate
problems. Token Binding is not designed to prevent replay of 0-RTT
data, although solutions for preventing the replay of Token Binding
might also be applicable to 0-RTT data.

6.1. Attacks on PSK-only Key Exchange and Token Binding

An attacker who possesses the PSK can eavesdrop on an existing
connection that uses that PSK to obtain a TokenBindingMessage that is
valid on the connection and then hijack the connection to send
whatever attacker-controlled data it wishes. Because the regular
exporter closes over the server random, this TokenBindingMessage is
valid only for that connection.

If the attacker does the same thing with a pure-PSK connection and
0-RTT Token Binding, the attacker can replay the original ClientHello
and the exporter will stay the same, allowing the attacker to obtain
a TokenBindingMessage from one connection and replay it on future
connections. The only way for a server to prevent this replay is to
prevent the client from ever repeating a client random in the
handshake.

If a server accepting connections with PSK-only key establishment is
concerned about the threat of PSK theft and also implements Token
Binding, then that server must either reject all 0-RTT token
bindings, or implement some form of preventing reuse of a client
random.

6.2. Exporter Replayability

The exporter specified in [I-D.ietf-tokbind-protocol] is chosen so
that a client and server have the same exporter value only if they
are on the same TLS connection. This prevents an attacker who can
read the plaintext of a TokenBindingMessage sent on that connection
from replaying that message on another connection (without also
having the token binding private key). The 0-RTT exporter only
covers the ClientHello and the PSK of the connection, so it does not
provide this guarantee.

An attacker with possession of the PSK secret and a transcript of the
ClientHello and early data sent by a client under that PSK can
extract the TokenBindingMessage, create a new connection to the
server (using the same ClientHello and PSK), and send different
application data with the same TokenBindingMessage. Note that the

ClientHello contains public values for the (EC)DHE key agreement that
is used as part of deriving the traffic keys for the TLS connection,
so if the attacker does not also have the corresponding private
values, they will not be able to read the server's response or send a
valid Finished message in the handshake for this TLS connection.
Nevertheless, by that point the server has already processed the
attacker's message with the replayed TokenBindingMessage.

This sort of replayability of a TokenBindingMessage is different than
the replayability caveat of 0-RTT application data in TLS 1.3. A
network observer can replay 0-RTT data from TLS 1.3 without knowing
any secrets of the client or server, but the application data that is
replayed is untouched. This replay is done by a more powerful
attacker who is able to view the plaintext and then spoof a
connection with the same parameters so that the replayed
TokenBindingMessage still validates when sent with different
application data.

6.3. Replay Mitigations

This section presents multiple ways that a client or server can
prevent the replay of a TokenBinding while still using Token Binding
with 0-RTT data.

If a client or server implements a measure that prevents all replays,
then its peer does not also need to implement such a mitigation. A
client that is concerned about replay SHOULD implement replay a
mitigation instead of relying solely on a signal from the server
through the replay indication extension.

6.3.1. Server Mitigations

If a server uses a session cache instead of stateless tickets, it can
enforce that a PSK generated for resumption can only be used once.
If an attacker tries to replay 0-RTT data (with a
TokenBindingMessage), the server will reject it because the PSK was
already used.

Preventing all replay of 0-RTT data is not necessary to prevent
replay of a TokenBinding. A server could implement a mechanism to
prevent a particular TokenBinding from being presented on more than
one connection. In cases where a server's TLS termination and
application layer processing happen in different locations, this
option might be easier to implement, especially when not all requests
have bound tokens. This processing can also take advantage of the
structure of the bound token, e.g. a token that identifies which user
is making a request could shard its store of which TokenBindings have
been seen based on the user ID.

   A server can prevent some, but not all, 0-RTT data replay with a
   tight time window for the ticket age that it will accept. See
   Section 6.4 for more details.

6.3.2. Client Mitigations

   A client cannot prevent a sufficiently motivated attacker from
   replaying a TokenBinding, but it can make it so difficult to replay
   the TokenBinding that it is easier for the attacker to steal the
   Token Binding key directly. If the client secures the resumption
   secret with the same level of protection as the Token Binding key,
   then the client has made it not worth the effort of the attacker to
   attempt to replay a TokenBinding. Ideally the resumption secret (and
   Token Binding key) are protected strongly and virtually non-
   exportable.

6.4. Early Data Ticket Age Window

   When an attacker with control of the PSK secret replays a
   TokenBindingMessage, it has to use the same ClientHello that the
   client used. The ClientHello includes an "obfuscated_ticket_age" in
   its EarlyDataIndication extension, which the server can use to narrow
   the window in which that ClientHello will be accepted. Even if a PSK
   is valid for a week, the server will only accept that particular
   ClientHello for a smaller time window based on the ticket age. A
   server should make their acceptance window for this value as small as
   practical to limit an attacker's ability to replay a ClientHello and
   send new application data with the stolen TokenBindingMessage.

6.5. Lack of Freshness

   The 0-RTT exporter value does not contain anything that the client
   cannot precompute before connecting to the server. Therefore, an
   attacker could have a client generate but not send a series of
   messages to take particular actions, and then selectively send one of
   those messages at a later date. If this attack includes deleting the
   resumption secret from the client, then these latent attacker-held
   messages will be the only ones to use that resumption secret and
   replay protections do not prevent this attack.

7. Acknowledgements

   The author would like to thank David Benjamin, Steven Valdez, Bill
   Cox, and Andrei Popov for their feedback and suggestions.

8. **Normative References**

   [I-D.ietf-tls-tls13]
               Rescorla, E., "The Transport Layer Security (TLS) Protocol
               Version 1.3", draft-ietf-tls-tls13-18 (work in progress),
               October 2016.

   [I-D.ietf-tokbind-negotiation]
               Popov, A., Nystrom, M., Balfanz, D., and A. Langley,
               "Transport Layer Security (TLS) Extension for Token
               Binding Protocol Negotiation", draft-ietf-tokbind-
               negotiation-05 (work in progress), September 2016.

   [I-D.ietf-tokbind-protocol]
               Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J.
               Hodges, "The Token Binding Protocol Version 1.0", draft-
               ietf-tokbind-protocol-10 (work in progress), September
               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>.

   [RFC5705]   Rescorla, E., "Keying Material Exporters for Transport
               Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705,
               March 2010, <http://www.rfc-editor.org/info/rfc5705>.

Author's Address

   Nick Harper
   Google Inc.

   Email: nharper@google.com