Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: July 15, 2017

B. Campbell
Ping Identity

January 11, 2017

HTTPS Token Binding and TLS Terminating Reverse Proxies
draft-campbell-tokbind-tls-term-00

Abstract

This document defines an HTTP header field that enables a TLS terminating reverse proxy to convey the information a backend server needs in order for it to process and validate a Token Binding Message sent by the client.

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 July 15, 2017.

Table of Contents

1. Introduction

Token Binding over HTTP [I-D.ietf-tokbind-https] provides a mechanism that enables HTTP servers to cryptographically bind cookies and other security tokens to TLS [RFC5246] connections. When Token Binding is negotiated in the TLS handshake [I-D.ietf-tokbind-negotiation] the client sends an encoded Token Binding Message [I-D.ietf-tokbind-protocol] as a header in each HTTP request, which proves possession of one or more private keys held by the client. The public portion of the keys are represented in the Token Binding IDs of the Token Binding Message and for each one there is a signature over some data, which includes the exported keying material [RFC5705] of the TLS connection. An HTTP server issuing cookies or other security tokens can associate them with the Token Binding ID, which ensures those tokens cannot be used successfully over a different TLS connection or by a different client than the one to which they were issued.

A fairly common deployment architecture for HTTPS applications is to have the backend HTTP application servers sit behind a reverse proxy that terminates TLS. The proxy is accessible to the internet and dispatches client requests to the appropriate backend server within a private network. The backend servers are not directly accessible outside the private network and are only reachable through the reverse proxy. The details of such deployments are typically opaque to clients who make requests to the proxy server and see responses as though they originated from the proxy server itself. TLS connections for HTTPS are established between each client and the reverse proxy server.

Token Binding facilitates a binding of security tokens to a key held by the client by way of the TLS connection between that client and the sever. In a TLS terminating reverse proxy deployment, however, the TLS connection is between the client and the proxy while the backend server is likely the system that will issue security tokens.

Additional steps are therefore needed to enable the use of Token Binding in such deployment architectures. In the absence of a standardized approach, different implementations will will address it differently, which will make interoperability between implementation difficult or impossible without complex configurations or custom integrations.

This document standardizes an HTTP header field named "Token-Binding-Context" that a TLS terminating reverse proxy adds to requests that it sends to the backend servers. The value of the header contains the information from its connection with the client that is necessary for the backend server to process and validate the Token Binding Message also in the request. The usage of the header, both the reverse proxy adding it and the application server using it rather than information from its inbound connection, are to be configuration options of the respective systems as they will not always be applicable.

1.1. Requirements Notation and Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. The Token-Binding-Context HTTP Header Field

When configured to do so, a reverse proxy that terminates TLS and negotiates Token Binding over HTTP [I-D.ietf-tokbind-https] with a client adds a "Token-Binding-Context" HTTP header field to the request that is dispatched to a backend server.

The "Token-Binding-Context" is a single HTTP header field-value as defined in Section 3.2 of [RFC7230], which MUST NOT have a list of values or occur multiple times in a request. The "Token-Binding-Context" header is only for use in HTTP requests and MUST NOT to be used in HTTP responses. The header field value is defined in ABNF [RFC5234] syntax as:

```
Token-Binding-Context = EncodedTBContextMessage
EncodedTBContextMessage = 47*( DIGIT / ALPHA /  "-" / "_" )

DIGIT = <Defined in Section B.1 of [RFC5234]>
ALPHA = <Defined in Section B.1 of [RFC5234]>
```

The header field name is "Token-Binding-Context" and its value is a base64url encoding of a Token Binding Context Message using the URL- and filename-safe character set described in Section 5 of [RFC4648],

with all trailing pad characters '=' omitted and without the inclusion of any line breaks, whitespace, or other additional characters.

The Token Binding Context Message is a byte sequence that contains the concatenation of the negotiated Token Binding Protocol Version and Key Parameters as well as the exported keying material (EKM) from the TLS connection between the client and reverse proxy. The first two bytes are the ProtocolVersion, as defined in Section 2 of [I-D.ietf-tokbind-negotiation], that the reverse proxy negotiated with the client. The third byte is the negotiated TokenBindingKeyParameters (also defined in Section 2 of [I-D.ietf-tokbind-negotiation]). The remaining 32 or more bytes are the EKM from the TLS connection between the client and the reverse proxy, as defined in Section 3.3 of [I-D.ietf-tokbind-protocol].

For example, below is an encoded Token Binding Context Message indicating version 1.0 of the protocol, ecdsap256(2) key parameters, and a 32 byte EKM:

```
AQACltcPRPoACC9N9lW5ESCvw4e6_6oISR38bwc2ddz7fFs4i
```

A backend server that receives a request from a trusted reverse proxy containing the "Token-Binding-Context" and "Sec-Token-Binding" headers decodes the Token Binding Context Message and uses its content to validate the encoded Token Binding Message as described in Section 2 of Token Binding over HTTP [I-D.ietf-tokbind-https] in place of information that otherwise would have come from the TLS connection.

Reverse proxies MUST only add the "Token-Binding-Context" header when explicitly configured to do so and MUST only dispatch requests containing it to trusted backend servers. Any occurrence of the "Token-Binding-Context" header in the request from the client MUST be removed or overwritten before forwarding the request. Backend servers MUST only accept the "Token-Binding-Context" header when explicitly configured to do so and only from trusted reverse proxies.

Forward proxies and other intermediaries MUST NOT add the "Token-Binding-Context" header to requests.

3. Security Considerations

The "Token-Binding-Context" header enables a reverse proxy and backend server to function together as though they are single logical deployment of HTTPS Token Binding. Use of the header outside that intended use case, however, may undermine the protections afforded by

Token Binding. Therefore steps must be taken to prevent unintended use, both in sending the header and in relying on its value.

Producing and consuming the "Token-Binding-Context" header should be a configurable option, respectively, in a reverse proxy and backend server (or individual application in that server). The default configuration for both should be to not use the "Token-Binding-Context" header thus requiring an "opt-in" to its usage.

Reverse proxies should only add the header to requests that are forwarded to trusted backend servers. Otherwise a legitimate EKM value might be disclosed to an unintended party.

Backend servers should only accept the header from trusted reverse proxies. And reverse proxies need to sanitize the incoming request before forwarding it on by removing or overwriting any existing instances of the "Token-Binding-Context" header. Otherwise arbitrary clients can control the EKM value as seen and used by the backend server.

The communication between a reverse proxy and backend server needs to be secured against eavesdropping and modification by unintended parties.

The configuration options and request sanitization are necessarily functionally of the respective servers. The other requirements can be met in a number of ways, which will vary based on specific deployments. The communication between a reverse proxy and backend server, for example, might be over a mutually authenticated TLS with the insertion and consumption of the "Token-Binding-Context" header occurring only on for that connection. Alternatively the network topology might dictate a private network such that the backend application is only able to accept requests from the reverse proxy and the proxy can only make requests to that server. Other deployments that meet the requirements set forth herein are also possible.

4. IANA Considerations

This document specifies the "Token-Binding-Context" HTTP header field, registration of which is requested in the "Permanent Message Header Field Names" registry defined in [RFC3864].

- Header Field Name: "Token-Binding-Context"
- Applicable protocol: http
- Status: standard
- Author/change Controller: IETF
- Specification Document(s): Section 2 of [[this specification]]

5. Normative References

[I-D.ietf-tokbind-https]
Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J. Hodges, "Token Binding over HTTP", draft-ietf-tokbind-https-07 (work in progress), November 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-06 (work in progress), November 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-11 (work in progress), November 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>.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <http://www.rfc-editor.org/info/rfc3864>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[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>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

Appendix A. Acknowledgements

The author would like to thank the following people for their contributions to the specification: Dirk Balfanz, John Bradley, Subodh Iyengar, Andrei Popov, Martin Thomson and others (please let me know, if you've contributed and I've forgotten you).

Appendix B. Open Issues

- might need this...

Appendix C. Document History

[[to be removed by the RFC Editor before publication as an RFC]]

draft-campbell-tokbind-tls-term-00

- Initial draft based on 'consensus to work on the problem' at the Seoul meeting. Slides and minutes from the meeting, respectively: https://www.ietf.org/proceedings/97/slides/slides-97-tokbind-reverse-proxies-00.pdf https://www.ietf.org/proceedings/97/minutes/minutes-97-tokbind-01.txt

Author's Address

Brian Campbell
Ping Identity

Email: brian.d.campbell@gmail.com

Token Binding Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 30, 2017

N. Harper
Google Inc.

June 28, 2017

Token Binding for 0-RTT TLS 1.3 Connections
draft-ietf-tokbind-tls13-0rtt-02

Abstract

This document describes how Token Binding can be used in the 0-RTT data of a TLS 1.3 connection. This involves a new TLS extension to negotiate and indicate the use of Token Binding in 0-RTT data. 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 December 30, 2017.

Copyright Notice

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.

Using Token Binding in 0-RTT data involves two main changes to Token Binding. The first is the use of a new TLS extension

"early_token_binding" to indicate whether a TLS session ticket can be used with Token Binding in 0-RTT data, and to indicate whether an attempted 0-RTT connection is using Token Binding in 0-RTT data. The second change is one that applies only if Token Binding in 0-RTT data is in use, which changes the definition of the TokenBinding.signature field to use TLS 1.3's early_exporter_secret.

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].

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. 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.

When early_exporter_secret is used as the Secret, the client MUST indicate this use so the server knows which secret to use in signature verification. This indication is done through a new Token Binding extension, "early_exporter" (with extension type TBD). This extension always has 0-length data, so the full Extension struct is the bytes {0xTBD, 0x00, 0x00}. The early_exporter extension MUST be present in every TokenBinding struct where the exporter that is signed uses the early_exporter_secret, and it MUST NOT be present in any other TokenBinding structs.

2.1. Selecting Which Exporter Secret to Use

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].

When a client sends a TokenBindingMessage in 0-RTT data, it must use the early_exporter_secret. If the server accepts the 0-RTT data, the client must continue to use the early_exporter_secret for the rest of the connection. If the server rejects 0-RTT data, the client must use the exporter_secret.

3. Negotiation

3.1. Indicating Use of 0-RTT Token Binding

The TLS extension "early_token_binding" (extension type TBD) is used in the TLS ClientHello and EncryptedExtensions to indicate use of 0-RTT Token Binding on the current connection. It is also used in a NewSessionTicket message to indicate that 0-RTT Token Binding may be used on a connection resumed with that ticket. In all cases, the "extension_data" field of this extension is empty, so the entire encoding of this extension is 0xTBD 0xTBD 0x00 0x00.

3.2. Token Binding Negotiation TLS Extension

In TLS 1.3, the "token_binding" extension is sent by a server in EncryptedExtensions, whereas in previous versions of TLS this extension was sent in the ServerHello message. On a 1-RTT connection (whether it be a new connection or resumption), no application data is sent in either direction before the "token_binding" TLS extension in the EncryptedExtensions, and the choice of Token Binding version and key parameter is up to the server based on what the client sent and what the server's preferences are, following the same processing rules as in [I-D.ietf-tokbind-negotiation].

3.3. Client Processing Rules

A client that supports Token Binding in 0-RTT data and receives a NewSessionTicket containing the "early_token_binding" extension must store with the ticket the Token Binding version and key parameter of the connection in which the ticket was issued.

A client that wishes to send a Token Binding message in 0-RTT data may only do so if the TLS connection in which the 0-RTT data is being sent is being resumed from a ticket which included the "early_token_binding" extension. Assuming the ticket included this extension, the client sends a ClientHello containing the "token_binding" extension, "early_data" extension, and "early_token_binding" extensions. The client must include in its "psk_key_exchange_modes" extension psk_dhe_ke.

The contents of the "token_binding" extension SHOULD be the same as they would be on a connection without "early_token_binding" to allow for the client and server to negotiate new Token Binding parameters if the early data is rejected. The Token Binding message sent in the 0-RTT data MUST be sent assuming that the same Token Binding version and key parameter from the connection where the ticket was received will also be negotiated on this connection. If the server includes the "early_data" extension in EncryptedExtensions in response to a

ClientHello with "early_token_binding", but the server does not include "early_token_binding" in EncryptedExtensions, or if the server's "token_binding" extension does not match the values of the connection where the ticket was received, then the client MUST terminate the TLS connection with an illegal_parameter alert.

It is valid for a client to send a ClientHello that contains both the "early_data" and "token_binding" extensions, but without the "early_token_binding" extension. This combination means that the client is attempting to resume a connection and is sending early data, but the client is not using Token Binding on this resumed connection (if the server accepts the early data). The presence of the "token_binding" extension is so the client can negotiate the use of Token Binding for this connection if the server rejects early data.

3.4. Server Processing Rules

When a server issues a NewSessionTicket on a connection where Token Binding was negotiated, and the NewSessionTicket includes an "early_data" extension indicating that the ticket may be used to send 0-RTT data, the server may also include the "early_token_binding" extension in the NewSessionTicket to indicate that this ticket can be used for a future connection with Token Binding in 0-RTT data. If the server includes the "early_token_binding" extension in the NewSessionTicket, the server MUST store with the ticket the Token Binding version and key parameter used for the connection in which the ticket was issued. The "early_token_binding" extension can appear in a NewSessionTicket message only if the "early_data" extension also appears in that message.

If a server receives a ClientHello with the "early_token_binding" extension and supports Token Binding in 0-RTT data, it MUST perform the following checks:

- If either the "early_data" or "token_binding" extensions are missing from the ClientHello, terminate the TLS connection with an illegal_parameter alert.

- If the ticket used for resumption is missing either of the "early_data" or "early_token_binding" extensions, reject the early data.

- Process the "token_binding" extension as if it were received on a 1-RTT connection and compute the Token Binding version and key parameter to use. If either of these values do not match the values that were negotiated on the connection where the ticket used for resumption was sent, reject the early data.

- Perform any other checks to decide whether to accept early data. If the server chooses to accept early data, include in EncryptedExtensions the "early_data" extension, "early_token_binding" extension, and "token_binding" extension with the same version and key parameter from the previous connection.

If a server accepts early data on a connection where "early_token_binding" was offered, it MUST use PSK with (EC)DHE key establishment.

The "early_token_binding" extension must be present in EncryptedExtensions exactly when both "early_data" and "token_binding" are present. A server that receives a ClientHello with "early_token_binding" cannot reject Token Binding and also accept early data at the same time. Said server may reject early data but still negotiate Token Binding.

A server might receive a ClientHello that includes both the "early_data" and "token_binding" extensions, but no "early_token_binding" extension. In this case, the server has three options:

1. Accept early data and continue the connection with no Token Binding,

2. Reject early data and negotiate the use of Token Binding for this connection, or

3. Reject early data and do not negotiate Token Binding for this connection.

The behavior for the "token_binding" extension in 0-RTT is similar to that of ALPN and SNI: the client predicts the result of the negotiation, and if the actual negotiation differs, the server rejects the early data.

4. Implementation Considerations

4.1. Not Implementing Token Binding on 0-RTT Connections

This spec has been designed so that both clients and servers can support Token Binding on some connections and 0-RTT data on other connections without needing to support Token Binding on 0-RTT connections.

A client that wishes to support both without supporting Token Binding on 0-RTT connections can function by completely ignoring the

"early_token_binding" TLS extension. When resuming a connection with early data, the client can still advertise support for Token Binding, providing the server the opportunity to accept early data (without Token Binding) or to reject early data and negotiate Token Binding. By always including the "token_binding" extension in its ClientHello, the client can prioritize Token Binding over 0-RTT.

A server can support both Token Binding and 0-RTT data without supporting Token Binding on 0-RTT connections by never minting NewSessionTickets containing the "early_token_binding" extension. Such a server that never mints NewSessionTickets with "early_token_binding" can ignore that extension in a ClientHello as it would only appear if the client is not spec compliant. On connections where a server negotiates Token Binding, the server SHOULD NOT include the "early_data" extension in a NewSessionTicket.

4.2. Adding Support for Token Binding on 0-RTT Connections

A server that supports early data but not Token Binding may wish to add support for Token Binding (and Token Binding on 0-RTT connections) at a later time. For a client to learn that a server supports Token Binding, the server must reject early data to send the "token_binding" extension.

4.3. 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.

5. IANA Considerations

This document defines a new TLS extension "early_token_binding" with code point TBD which needs to be added to IANA’s TLS "ExtensionType Values" registry.

This document defines a new Token Binding extension "early_exporter", which needs to be added to the IANA "Token Binding 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. Proof of Possession of Token Binding Key

When a Token Binding signature is generated using the exporter with early_exporter_secret, the value being signed is under the client's control. An attacker with temporary access to the Token Binding private key can generate Token Binding signatures for as many future connections as it has NewSessionTickets for. An attacker can construct these to be usable at any time in the future up until the NewSessionTicket's expiration. Section 4.6.1 of [I-D.ietf-tls-tls13] requires that a NewSessionTicket be valid for a maximum of 7 days.

Unlike in [I-D.ietf-tokbind-protocol], where the proof of possession of the Token Binding key proves that the client had possession at the time the TLS handshake finished, 0-RTT Token Binding only proves that the client had possession of the Token Binding key at some point after receiving the NewSessionTicket used for that connection.

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 mitigate the replay of a TokenBinding while still using Token Binding with 0-RTT data. Note that even with replay mitigations, 0-RTT Token Binding is vulnerable to other attacks.

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.

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-tls13-20 (work in progress), April 2017.

[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-08 (work in progress), April 2017.

[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-14 (work in progress), April 2017.

[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

Token Binding Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 28, 2019

G. Mandyam
Qualcomm Technologies Inc.
L. Lundblade
Security Theory LLC
J. Azen
Qualcomm Technologies Inc.

January 24, 2019

Attested TLS Token Binding
draft-mandyam-tokbind-attest-07

Abstract

Token binding allows HTTP servers to bind bearer tokens to TLS connections. In order to do this, clients or user agents must prove possession of a private key. However, proof-of-possession of a private key becomes truly meaningful to a server when accompanied by an attestation statement. This specification describes extensions to the existing token binding protocol to allow for attestation statements to be sent along with the related token binding messages.

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 https://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 July 28, 2019.

Copyright Notice

Table of Contents

1. Introduction

[RFC8471] and [RFC8472] describe a framework whereby servers can leverage cryptographically-bound authentication tokens in part to create uniquely-identifiable TLS bindings that can span multiple connections between a client and a server. Once the use of token binding is negotiated as part of the TLS handshake, an application layer message (the Token Binding message) may be sent from the client to the relying party whose primary purpose is to encapsulate a signature over a value associated with the current TLS session. The payload used for the signature is the token binding public key (see [RFC8471]). Use of the token binding public key allows for generation of the attestation signature once over the lifetime of the public key.

Proof-of-possession of a private key is useful to a relying party, but the associated signature in the Token Binding message does not provide an indication as to how the private key is stored and in what kind of environment the associated cryptographic operation takes place. This information may be required by a relying party in order

to satisfy requirements regarding client platform integrity. Therefore, attestations are sometimes required by relying parties in order for them to accept signatures from clients. As per the definition in [I-D.birkholz-tuda], "remote attestation describes the attempt to determine the integrity and trustworthiness of an endpoint -- the attestee -- over a network to another endpoint -- the verifier -- without direct access." Attestation statements are therefore widely used in any server verification operation that leverages client cryptography.

TLS token binding can therefore be enhanced with remote attestation statements. The attestation statement can be used to augment Token Binding message. This could be used by a relying party for several different purpose, including (1) to determine whether to accept token binding messages from the associated client, or (2) require an additional mechanism for binding the TLS connection to an authentication operation by the client.

2. Attestation Enhancement to TLS Token Binding Message

The attestation statement can be processed 'in-band' as part of the Token Binding Message itself. This document leverages the TokenBinding.extensions field of the Token Binding Message as described in Section 3.4 of [RFC8471], where the extension data conforms to the guidelines of Section 6.3 of the same document. The value of the extension, as required by this same section, is assigned per attestation type. The extension data takes the form of a CBOR (compact binary object representation) Data Definition Language construct, i.e. CDDL.

```
extension_data = {attestation}
attestation = (
  attestation_type:  tstr,
  attestation_data:  bstr,
  )
```

The attestation data is determined according to the attestation type. In this document, the following types are defined: "KeyStore" (where the corresponding attestation data defined in [Keystore]) and "TPMv2" (where the corresponding attestation data defined in [TPMv2]). Additional attestation types may be accepted by the token binding implementation (for instance, see Section 8 of [webauthn]).

The attestation data will likely include a signature over a challenge (depenting on the attestation type). The challenge can be used to prevent replay of the attestation. However since the attestation is

itself part of the token binding message (which has its own anti-replay protection mechanism), the attestation signature need only be generated over a known payload associated with the TLS token binding session - the token binding public key. As a result, the token binding client only needs to send the attestation once during the lifetime of the token binding public key. In other words, if an attestation is included in the token binding message, it should only be sent in the initial token binding message following the creation of the token binding key pair.

2.1. KeyStore Attestation

KeyStore attestation is relevant to the Android operating system. The Android Keystore mechanism allows for an application (such as a browser implementing the Token Binding stack) to create a key pair, export the public key, and protect the private key in a hardware-backed keystore. The Android Keystore can then be used to verify a keypair using the Keystore Attestation mechanism, which involves signing a payload according to a public key that chains to a root certificate signed by an attestation root key that is specific to the device manufacturer.

The octet value of the token binding extension that serves as identifiaction for the Keystore attestation type is requested to be 0.

KeyStore attestation provides a signature over a payload generated by the application. The payload is a SHA-256 hash of the token binding public key corresponding to the current TLS connection (see Section 3.3 of [RFC8471]). Then the attestation takes the form of a signature, a signature-generation algorithmic identifier corresponding to the COSE algorithm registry ([cose_iana]), and a chain of DER-encoded x.509 certificates:

```
attestation_data = (
  alg: int,
  sig:  bytes,
  x5c: [credCert: bytes, *(caCert: bytes)]
  )
```

2.1.1. Verification Procedures

The steps at the server for verifying a Token Binding KeyStore Attestation are:

- Retrieve token binding public key for the current TLS connection, and compute is SHA-256 hash.
- Verify that attestation_data is in the expected CBOR format.
- Parse the first certificate listed in x5c and extract the public key and challenge. If the challenge does not match the SHA-256 hash of the token binding public key then the attestation is invalid.
- If the challenge matches the expected hash of the token binding public key, verify the sig with respect to the extracted public key and algorithm from the previous step.
- Verify the rest of the certificate chain up to the root. The root certificate must match the expected root for the device.

2.2. TPMv2 Attestation

Version 2 of the Trusted Computing Group's Trusted Platform Module (TPM) specification provides for an attestation generated within the context of a TPM. The attestation then is defined as

```
attestation_data = (
  alg: int,
  tpmt_sig:  bytes,
  tpms_attest:  bytes,
  x5c: [credCert: bytes, *(caCert: bytes)]
  )
```

The tpmt_sig is generated over a tpms_attest structure signed with respect to the certificate chain provided in the x5c array, and the algorithmic identifier corresponding to the COSE algorithm registry ([cose_iana]). It is derived from the TPMT_SIGNATURE data structure defined in Section 11.3.4 of [TPMv2]. tpms_attest is derived from the TPMS_ATTEST data structure in Section 10.2.8 of [TPMv2], specifically with the extraData field being set to a SHA-256 hash of the token binding public key.

The octet value of the token binding extension that serves as identifiaction for the TPMv2 attestation type is requested to be 1.

2.2.1. Verification Procedures

The steps for verifying a Token Binding TPMv2 Attestation are:

- Extract the token binding public key for the current TLS connection.
- Verify that attestation_data is in the expected CBOR format.
- Parse the first certificate listed in x5c and extract the public key.
- Verify the tpms_attest structure,which includes
  - Verify that the type field is set to TPM_ST_ATTEST_CERTIFY.
  - Verify that extraData is equivalent to the SHA-256 hash of the token binding public key for the current TLS connection.
  - Verify that magic is set to the expected TPM_GENERATED_VALUE for the expected command sequence used to generate the attestation.
  - Verification of additonal TPMS_ATTEST data fields is optional.
- Verify tpmt_sig with respect to the public key provided in the first certifcate in x5c, using the algorithm as specified in the sigAlg field (see Sections 11.3.4, 11.2.1.5 and 9.29 of [TPMv2]).

3. Extension Support Negotiation

Even if the client supports a Token Binding extension, it may not be desirable to send the extension if the server does not support it. The benefits of client-suppression of an extension could include saving of bits "over the wire" or simplified processing of the Token Binding message at the server. Currently, extension support is not communicated as part of the Token Binding extensions to TLS (see [RFC8472]).

It is proposed that the Client and Server Hello extensions defined in Sections 3 and 4 of [RFC8472] be extended so that endpoints can communicate their support for specific TokenBinding.extensions. With reference to Section 3, it is recommended that the "token_binding" TLS extension be augmented by the client to include supported TokenBinding.extensions as follows:

```
enum {
    attestation(0), (255)
} TokenBindingExtensions;

struct {
    TB_ProtocolVersion token_binding_version;
    TokenBindingKeyParameters key_parameters_list<1..2^8-1>;
    TokenBindingExtensions supported_extensions_list<1..2^8-1>
} TokenBindingParameters;
```

The "supported_extensions_list" contains the list of identifiers of all token binding message extensions supported by the client. A server supporting token binding extensions will respond in the server hello with an appropriate "token_binding" extension that includes a "supported_extensions_list". This list must be a subset of the the extensions provided in the client hello.

Since a TLS extension cannot itself be extended, the "token_binding" TLS extension cannot be reused. Therefore it is proposed that a new TLS extension be defined - "token_binding_with_extensions". This TLS extension codepoint is identical to the existing "token_binding" extension except for the additional data structures defined above.

3.1. Negotiating Token Binding Protocol Extensions

The negotation described in Section 4 of [RFC8472] still applies, except now the "token_binding_with_extensions" codepoint would be used if the client supports any token binding extension. In addition, a client can receive a "supported_extensions_list" from the server as part of the server hello. The client must terminate the handshake if the "supported_extensions_list" received from the server is not a subset of the "supported_extensions_list" sent by the client in the client hello. If the server hello list of supported extensions is a subset of the client supported extensions, then the client must only send those extensions specified in the server hello in the Token Binding protocol. If the server hello does not include a "supported_extensions_list", then the client must not send any extensions along with the Token Binding Message.

4. Example - Platform Attestation for Anomaly Detection

An example of where a platform-based attestation is useful can be for remote attestation based on client traffic anomaly detection. Many network infrastructure deployments employ network traffic monitors for anomalous pattern detection. Examples of anomalous patterns detectable in the TLS handshake could be unexpected cipher suite negotiation for a given source/destination pairing. In this case, it

may be desirable for a client-enhanced attestation reflecting for instance that an expected offered cipher suite in the client hello message is present or the originating browser integrity is intact (e.g. through a hash over the browser application package). If the network traffic monitor can interpret the atttestation included in the token binding message, then it can verify the attestation and potentially emit alerts based on an unexpected attestation.

5. IANA Considerations

This memo includes the following requests to IANA.

5.1. TLS Extensions Registry

This document proposes an update of the TLS "ExtensionType Values" registry. The following addition to the registry is requested:

Value: TBD

Extension name: token_binding_with_extensions

Reference: this document

Recommended: Yes

5.2. Token Binding Extensions for Attestation

This document proposes two extensions conformant with Section 6.3 of [RFC8471], with the following specifics:

Androoid Keystore Attestation:

- Value: 0
- Description: Android Keystore Attestation
- Specification: This document

TPM v2 Attestation:

- Value: 1
- Description: TPMv2 Attestation
- Specification: This document

6. Security and Privacy Considerations

The security and privacy considerations provided in Section 7 of [RFC8471] are applicable to the attestation extensions proposed in this document. Additional considerations are provided in this section.

6.1. Attestation Privacy Considerations

The root signing key for the certificate chain used in verifying an attestation can be unique to the device. As a result, this can be used to track a device and/or end user. This potential privacy issue can be mitigated by the use of batch keys as an alternative to unique keys, or by generation of origin-specific attestation keys.

The attestation data may also contain device-specific identifiers, or information that can be used to fingerprint a device. Sensitive information can be excluded from the attestation data when this is a concern.

7. Acknowledgments

Thanks to Andrei Popov for his detailed review and recommendations.

8. References

8.1. Normative References

[cose_iana]
Internet Assigned Numbers Authority, "COSE Algorithms", <https://www.iana.org/assignments/cose/cose.xhtml#algorithms>.

[I-D.greevenbosch-appsawg-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-11 (work in progress), July 2017.

[Keystore]
Google Inc., "Verifying hardware-backed key pairs with Key Attestation", <https://developer.android.com/training/articles/security-key-attestation>.

[RFC8471] Popov, A., Ed., Nystroem, M., Balfanz, D., and J. Hodges, "The Token Binding Protocol Version 1.0", RFC 8471, DOI 10.17487/RFC8471, October 2018, <https://www.rfc-editor.org/info/rfc8471>.

[RFC8472] Popov, A., Ed., Nystroem, M., and D. Balfanz, "Transport Layer Security (TLS) Extension for Token Binding Protocol Negotiation", RFC 8472, DOI 10.17487/RFC8472, October 2018, <https://www.rfc-editor.org/info/rfc8472>.

[RFC8473] Popov, A., Nystroem, M., Balfanz, D., Ed., Harper, N., and J. Hodges, "Token Binding over HTTP", RFC 8473, DOI 10.17487/RFC8473, October 2018, <https://www.rfc-editor.org/info/rfc8473>.

[TPMv2] The Trusted Computing Group, "Trusted Platform Module Library, Part 2: Structures", September 2016, <http://www.trustedcomputinggroup.org/wp-content/uploads/TPM-Rev-2.0-Part-2-Structures-01.38.pdf>.

[webauthn] The Worldwide Web Consortium, "Web Authentication: An API for accessing Scoped Credentials", <https://www.w3.org/TR/webauthn/>.

8.2. Informative References

[I-D.birkholz-tuda] Fuchs, A., Birkholz, H., McDonald, I., and C. Bormann, "Time-Based Uni-Directional Attestation", draft-birkholz-tuda-02 (work in progress), July 2016.

Authors' Addresses

Giridhar Mandyam
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California 92121
USA

Phone: +1 858 651 7200
Email: mandyam@qti.qualcomm.com

Laurence Lundblade
Security Theory LLC

Email: lgl@island-resort.com

Jon Azen
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California 92121
USA

Phone: +1 858 651 9476
Email: jazen@qti.qualcomm.com