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: September 14, 2017

N. Harper
Google Inc.

March 13, 2017

Token Binding for 0-RTT TLS 1.3 Connections
draft-ietf-tokbind-tls13-0rtt-01

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 September 14, 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.

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 either early_exporter_secret or exporter_secret. 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.

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.

2.1.1. 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. After the client receives an application-layer response from the server, it must use the exporter_secret for all future token bindings on that connection. Requests sent after

the client's TLS Finished message, but before the client processes any application-layer response from the server, may use either exporter secret in their token bindings.

A server may choose to reject an application message containing a Token Binding that uses the early_exporter_secret. If it chooses to do so, it may send an application message indicating that the client should re-send the request (with a new Token Binding). In HTTP, this could be done with a 307 status code.

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 5.4.1) 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. 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.

This document defines a new Token Binding extension "early_exporter", which needs to be added to the IANA "Token Binding Extensions" registry.

5. 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.

5.1. 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.

5.2. 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.

5.3. 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.

5.4. 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 a replay mitigation instead of relying solely on a signal from the server

through the replay indication extension. Note that even with replay mitigations, 0-RTT Token Binding is vulnerable to other attacks.

5.4.1. 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 5.5 for more details.

5.4.2. 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.

5.5. 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. Acknowledgements

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

7. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-19 (work in progress), March 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-07 (work in progress), February 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-13 (work in progress), February 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: September 8, 2017

G. Mandyam
L. Lundblade
J. Azen
Qualcomm Technologies Inc.

March 7, 2017

Attested TLS Token Binding
draft-mandyam-tokbind-attest-01

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 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 September 8, 2017.

Copyright Notice

Table of Contents

1. Introduction

[I-D.ietf-tokbind-protocol] and [I-D.ietf-tokbind-negotiation] describe a framework whereby servers can leverage cryptographically-bound authentication tokens to verify TLS connections. This is useful for prevention of man-in-the-middle attacks on TLS sessions, and provides a mechanism by which identity federation systems can be leveraged by a relying party to verify a client based on proof-of-possession of a private key.

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 (Exported Key Material, i.e. EKM - see [I-D.ietf-tokbind-protocol]).

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 [I-D.ietf-tokbind-protocol], where the extension data conforms to the guidelines of Section 6.3 of the same document. 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: "packed" (where the corresponding attestation data defined in [Webauthn]) and "TPM" (where the corresponding attestation data defined in [TPM]). Additional attestation types may be accepted by the token binding implementation.

3. 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.

4. IANA Considerations

This memo includes no request to IANA.

5. References

5.1. Normative References

[I-D.greevenbosch-appsawg-cbor-cddl]
Vigano, C. and H. Birkholz, "CBOR data definition language (CDDL): a notational convention to express CBOR data structures", draft-greevenbosch-appsawg-cbor-cddl-09 (work in progress), September 2016.

[I-D.ietf-tokbind-https]
Popov, A., Nystrom, M., Balfanz, D., Langley, A., and J. Hodges, "Token Binding over HTTP", draft-ietf-tokbind-https-05 (work in progress), July 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-03 (work in progress), July 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-08 (work in progress), July 2016.

[TPM] The Trusted Computing Group, "Trusted Platform Module Library, Part 1: Architecture", October 2014.

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

5.2. 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
Qualcomm Technologies Inc.
5775 Morehouse Drive
San Diego, California 92121
USA

Phone: +1 858 658 3584
Email: llundbla@qti.qualcomm.com

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

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