Attested TLS Token Binding
draft-mandyam-tokbind-attest-04

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 December 13, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must
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..."
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.

```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]).

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.

KeyStore attestation provides a signature over a payload generated by
the application. Since in this case the application is the Token
Binding stack resident on the device, the payload is the Exported Key
Material (EKM) corresponding to the current TLS connection (see
Section 3.3 of [I-D.ietf-tokbind-protocol]). Then the attestation
takes the form of a signature accompanies by a chain of DER-encoded
x.509 certificates:

\[
\text{attestation\_data = (}
\begin{array}{l}
\text{sig: bytes,} \\
\text{x5c: [credCert: bytes, *(caCert: bytes)]}
\end{array}
\)
\]

2.1.1. Verification Procedures

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

- Extract EKM for 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, algorithm and challenge. If the challenge does not match the
  EKM then the attestation is invalid.
- If the challenge matches the EKM, 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 = (  
    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. 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 EKM.

2.2.1. Verification Procedures

The steps for verifying a Token Binding TPMv2 Attestation are:

- Extract EKM for 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 EKM.
  * Verify that magic is set to the expected TPM_GENERATED_VALUE for the expected command sequence used to generate the attestation.
  * Verification of additional TPMS_ATTEST data fields is optional.
- Verify tpmt_sig with respect to the public key provided in the first certificate 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 [I-D.ietf-tokbind-negotiation]).

It is proposed that the Client and Server Hello extensions defined in Sections 3 and 4 of [I-D.ietf-tokbind-negotiation] 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:

```c
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.

### 3.1. Negotiating Token Binding Protocol Extensions

The negotiation described in Section 4 of [I-D.ietf-tokbind-negotiation] still applies. 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 attestation 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 no request to IANA.

6. References

6.1. Normative References

[I-D.greevenbosch-appsawg-cbor-cddl]

[I-D.ietf-tokbind-https]

[I-D.ietf-tokbind-negotiation]

[I-D.ietf-tokbind-protocol]
6.2. Informative References

[I-D.birkholz-tuda]

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