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

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:

- o Extract the token binding public key for the current TLS connection.

- o Verify that attestation_data is in the expected CBOR format.

- o Parse the first certificate listed in x5c and extract the public key.

- o 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.

- o 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