TLS
Internet-Draft
Intended status: Standards Track
Expires: September 6, 2018

N. Sullivan
Cloudflare Inc.

March 05, 2018

Exported Authenticators in TLS
draft-ietf-tls-exported-authenticator-06

Abstract

This document describes a mechanism in Transport Layer Security (TLS) to provide an exportable proof of ownership of a certificate that can be transmitted out of band and verified by the other party.

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 6, 2018.

Table of Contents

1. Introduction

This document provides a way to authenticate one party of a Transport Layer Security (TLS) communication to another using a certificate after the session has been established. This allows both the client and server to prove ownership of additional identities at any time after the handshake has completed. This proof of authentication can be exported and transmitted out of band from one party to be validated by the other party.

This mechanism provides two advantages over the authentication that TLS natively provides:

multiple identities - Endpoints that are authoritative for multiple identities - but do not have a single certificate that includes all of the identities - can authenticate with those identities over a single connection.

spontaneous authentication - Endpoints can authenticate after a connection is established, in response to events in a higher-layer protocol, as well as integrating more context.

This document intends to replace much of the functionality of renegotiation in previous versions of TLS. It has the advantages over renegotiation of not requiring additional on-the-wire changes during a connection. For simplicity, only TLS 1.2 and later are supported.

Post-handshake authentication is defined in TLS 1.3, but it has the disadvantage of requiring additional state to be stored in the TLS state machine and it composes poorly with multiplexed connection protocols like HTTP/2. It is also only available for client authentication. This mechanism is intended to be used as part of a replacement for post-handshake authentication in applications.

2. Conventions and Terminology

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. Authenticator Request

The authenticator request is a structured message that can be exported from either party of a TLS connection. It can be transmitted to the other party of the TLS connection at the application layer. The application layer protocol used to send the authenticator SHOULD use TLS as its underlying transport to keep the request confidential.

An authenticator request message can be constructed by either the client or the server. This authenticator request uses the CertificateRequest message structure from Section 4.3.2 of [TLS13]. This message does not include the TLS record layer and is therefore not encrypted with a handshake key.

The CertificateRequest is used to define the parameters in a request for an authenticator. The definition for TLS 1.3 is:

```
struct {
   opaque certificate_request_context<0..2^8-1>;
   Extension extensions<2..2^16-1>;
} CertificateRequest;
```

certificate_request_context: An opaque string which identifies the certificate request and which will be echoed in the authenticator message. The certificate_request_context MUST be unique within the scope of this connection (thus preventing replay of

authenticators). The certificate_request_context SHOULD be chosen to be unpredictable to the peer (e.g., by randomly generating it) in order to prevent an attacker who has temporary access to the peer's private key from pre-computing valid authenticators.

extensions: The extensions that are allowed in this structure include the extensions defined for CertificateRequest messages defined in Section 4.2. of [TLS13] and the server_name [RFC6066] extension, which is allowed for client-generated authenticator requests.

4. Authenticator

The authenticator is a structured message that can be exported from either party of a TLS connection. It can be transmitted to the other party of the TLS connection at the application layer. The application layer protocol used to send the authenticator SHOULD use TLS as its underlying transport to keep the certificate confidential.

An authenticator message can be constructed by either the client or the server given an established TLS connection, a certificate, and a corresponding private key. For clients, an authenticator request is required; for servers an authenticator request is optional. The authenticator uses the message structures from Section 4.4 of [TLS13], but different parameters. These messages do not include the TLS record layer and are therefore not encrypted with a handshake key.

4.1. Authenticator Keys

Each authenticator is computed using a Handshake Context and Finished MAC Key derived from the TLS session. These values are derived using an exporter as described in [RFC5705] (for TLS 1.2) or [TLS13] (for TLS 1.3). These values use different labels depending on the role of the sender:

- The Handshake Context is an exporter value that is derived using the label "EXPORTER-client authenticator handshake context" or "EXPORTER-server authenticator handshake context" for authenticators sent by the client and server respectively.

- The Finished MAC Key is an exporter value derived using the label "EXPORTER-client authenticator finished key" or "EXPORTER-server authenticator finished key" for authenticators sent by the client and server respectively.

The context_value used for the exporter is absent (length zero) for all four values. The length of the exported value is equal to the

length of the output of the hash function selected in TLS for the pseudorandom function (PRF). Cipher suites that do not use the TLS PRF MUST define a hash function that can be used for this purpose or they cannot be used.

If the connection is TLS 1.2, the master secret MUST have been computed with the extended master secret [RFC7627] to avoid key synchronization attacks.

4.2. Authenticator Construction

An authenticator is formed from the concatenation of TLS 1.3 [TLS13] Certificate, CertificateVerify, and Finished messages.

If an authenticator request is present, the extensions used to guide the construction of these messages are taken from the authenticator request. If there is no authenticator request, the extensions are chosen from the TLS handshake. That is, the extensions received in a ClientHello (for servers), ServerHello (for clients in TLS 1.2), or EncryptedExtensions (for clients in TLS 1.3).

4.2.1. Certificate

The certificate to be used for authentication and any supporting certificates in the chain. This structure is defined in [TLS13], Section 4.4.2.

The certificate message contains an opaque string called certificate_request_context, which is extracted from the authenticator request if present. If no authenticator request is provided, the certificate_request_context can be chosen arbitrarily.

The certificates chosen in the Certificate message MUST conform to the requirements of a Certificate message in the negotiated version of TLS. In particular, the certificate MUST be valid for the a signature algorithm indicated by the peer in a "signature_algorithms" extension, as described in Section 4.2.3 of [TLS13] and Sections 7.4.2 and 7.4.6 of [RFC5246].

In addition to "signature_algorithms", the "server_name" [RFC6066], "certificate_authorities" (Section 4.2.4. of [TLS13]), or "oid_filters" (Section 4.2.5. of [TLS13]) extensions are used to guide certificate selection. These extensions are taken from the authenticator request if present, or the TLS handshake if not.

Alternative certificate formats such as [RFC7250] Raw Public Keys are not supported in this version of the specification.

If an authenticator request was provided, the Certificate message MUST contain only extensions present in the authenticator request. Otherwise, the Certificate message MUST contain only extensions present in the TLS handshake.

4.2.2. CertificateVerify

This message is used to provide explicit proof that an endpoint possesses the private key corresponding to its certificate. The definition for TLS 1.3 is:

```
struct {
   SignatureScheme algorithm;
   opaque signature<0..2^16-1>;
} CertificateVerify;
```

The algorithm field specifies the signature algorithm used (see Section 4.2.3 of [TLS13] for the definition of this field). The signature is a digital signature using that algorithm.

The signature scheme MUST be a valid signature scheme for TLS 1.3. This excludes all RSASSA-PKCS1-v1_5 algorithms and combinations of ECDSA and hash algorithms that are not supported in TLS 1.3.

If an authenticator request is present, the signature algorithm MUST be chosen from one of the signature schemes in the authenticator request. Otherwise, the signature algorithm used should be chosen from the "signature_algorithms" extension sent by the peer in the TLS handshake.

The signature is computed using the over the concatenation of:

- A string that consists of octet 32 (0x20) repeated 64 times
- The context string "Exported Authenticator" (which is not NULL-terminated)
- A single 0 byte which serves as the separator
- The hashed authenticator transcript

The authenticator transcript is the hash of the concatenated Handshake Context, authenticator request (if present), and Certificate message:

"Hash(Handshake Context || authenticator request || Certificate) "

Where Hash is the hash function negotiated by TLS. If the authenticator request is not present, it is omitted from this construction (that is, it is zero length).

4.2.3. Finished

A HMAC [HMAC] over the hashed authenticator transcript, which is the concatenated Handshake Context, authenticator request (if present), Certificate, and CertificateVerify:

"Hash(Handshake Context || authenticator request || Certificate || CertificateVerify) "

The HMAC is computed using the same hash function using the Finished MAC Key as a key.

4.2.4. Authenticator Creation

An endpoint constructs an authenticator by serializing the Certificate, CertificateVerify, and Finished as TLS handshake messages and concatenating the octets:

"Certificate || CertificateVerify || Finished "

A given authenticator can be validated by checking the validity of the CertificateVerify message given the authenticator request (if used) and recomputing the Finished message to see if it matches.

5. API considerations

The creation and validation of both authenticator requests and authenticators SHOULD be implemented inside the TLS library even if it is possible to implement it at the application layer. TLS implementations supporting the use of exported authenticators MUST provide application programming interfaces by which clients and servers may request and verify exported authenticator messages.

Notwithstanding the success conditions described below, all APIs MUST fail if:

- the connection uses a TLS version of 1.1 or earlier, or
- the connection is TLS 1.2 and the extended master secret [RFC7627] was not used

The following sections describes APIs that are considered necessary to implement exported authenticators. These are informative only.

5.1. The "request" API

The "request" API takes as input:

- certificate_request_context (from 0 to 255 bytes)
- set of extensions to include (this MUST include signature_algorithms)

It returns an authenticator request, which is a sequence of octets that includes a CertificateRequest message.

5.2. The "get context" API

The "get context" API takes as input:

- authenticator

It returns the certificate_request_context.

5.3. The "authenticate" API

The "authenticate" takes as input:

- a set of certificate chains and associated extensions (OCSP, SCT, etc.)
- a signer (either the private key associated with the certificate, or interface to perform private key operation) for each chain
- an optional authenticator request or certificate_request_context (from 0 to 255 bytes)

It returns the exported authenticator as a sequence of octets. It is RECOMMENDED that the logic for selecting the certificates and extensions to include in the exporter is implemented in the TLS library. Implementing this in the TLS library lets the implementer take advantage of existing extension and certificate selection logic.

It is also possible to implement this API outside of the TLS library using TLS exporters. This may be preferable in cases where the application does not have access to a TLS library with these APIs or when TLS is handled independently of the application layer protocol.

5.4. The "validate" API

The "validate" API takes as input:

- an optional authenticator request
- an authenticator

It returns the certificate chain and extensions.

6. IANA Considerations

This document has no IANA actions.

7. Security Considerations

The Certificate/Verify/Finished pattern intentionally looks like the TLS 1.3 pattern which now has been analyzed several times. In the case where the client presents an authenticator to a server, [SIGMAC] presents a relevant framework for analysis.

Authenticators are independent and unidirectional. There is no explicit state change inside TLS when an authenticator is either created or validated.

- This property makes it difficult to formally prove that a server is jointly authoritative over multiple certificates, rather than individually authoritative over each.
- There is no indication in the TLS layer about which point in time an authenticator was computed. Any feedback about the time of creation or validation of the authenticator should be tracked as part of the application layer semantics if required.

The signatures generated with this API cover the context string "Exported Authenticator" and therefore cannot be transplanted into other protocols.

8. Acknowledgements

Comments on this proposal were provided by Martin Thomson. Suggestions for Section 7 were provided by Karthikeyan Bhargavan.

9. References

9.1. Normative References

[HMAC] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://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, <https://www.rfc-editor.org/info/rfc5705>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <https://www.rfc-editor.org/info/rfc7250>.

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <https://www.rfc-editor.org/info/rfc7627>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-26 (work in progress), March 2018.

9.2. Informative References

[SIGMAC] Krawczyk, H., "A Unilateral-to-Mutual Authentication Compiler for Key Exchange (with Applications to Client Authentication in TLS 1.3)", 2016, <https://eprint.iacr.org/2016/711.pdf>.

Author's Address

Nick Sullivan
Cloudflare Inc.

Email: nick@cloudflare.com