TLS
Internet-Draft
Intended status: Standards Track
Expires: February 6, 2017

N. Sullivan
CloudFlare Inc.
M. Thomson
Mozilla
M. Bishop
Microsoft

August 5, 2016

Post-Handshake Authentication in TLS
draft-sullivan-tls-post-handshake-auth-00

Abstract

This document describes a mechanism for performing post-handshake certificate-based authentication in Transport Layer Security (TLS) versions 1.3 and later. This includes both spontaneous and solicited authentication of both client and server.

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 February 6, 2017.

Table of Contents

1. Introduction

This document defines 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 solicit proof of ownership of additional identities at any time after the handshake has completed. It also allows for both the client and server to spontaneously provide a certificate and proof of ownership of the private key to the other party.

This mechanism is useful in the following situations:

- servers that have the ability to serve requests from multiple domains over the same connection but do not have a certificate that is simultaneously authoritative for all of them

- servers that have resources that require client authentication to access and need to request client authentication after the connection has started

- clients that want to assert their identity to a server after a connection has been established

- clients that want a server to re-prove ownership of their private key during a connection

- clients that wish to ask a server to authenticate for a new domain not covered by the certificate included in the initial handshake

This document intends to replace the use of renegotiation for changing the authentication of peers. It has an advantage over renegotiation in that it only takes at most one round trip and it does not include an additional key exchange.

This document describes spontaneous and solicited modes for both client and server authentication. Spontaneous authentication allows an endpoint to advertise a certificate without explicitly being requested. Solicited authentication allows an endpoint to request that its peer provide authentication details.

Support for different modes of authentication is negotiated using a new "post_handshake_auth" extension. New handshake messages are defined for use after completion of the initial handshake, these mirror the authentication messages that are used in the TLS 1.3 handshake.

2. Post-Handshake Authentication

There is a total of four different exchanges that are enabled by this specification. Solicited and spontaneous authentication exchanges are largely the same for both peers. This section describes how each exchange operates.

In all cases, a unique value for the certificate_request_context is chosen. This allows for identification of the authentication flow in application protocols that use TLS. Exchanges that are initiated by the client start with an octet that has the most significant bit set; exchanges initiated by the server have the most significant bit cleared.

2.1. Spontaneous Authentication

An endpoint that wishes to offer spontaneous authentication sends a Certificate, CertificateVerify, and Finished message.

```
  Certificate
  CertificateVerify
  Finished              ---------->
```

No application data records or any other handshake messages can be interleaved with these messages. An endpoint MUST abort a connection if it does not receive these messages in a contiguous sequence. A fatal "unexpected_message" alert SHOULD be sent if these messages do not appear in sequence.

A client MUST NOT initiate spontaneous authentication unless the server included client_auth_spontaneous in its "post_handshake_auth" extension. Similarly, a server MUST NOT initiate spontaneous authentication unless it included server_auth_solicited in its "post_handshake_auth" extension.

2.2. Solicited Authentication

Solicited authentication is initiated by sending a CertificateRequest message.

Endpoints that request that their peer authenticate need to account for delays in processing requests. In particular, client authentication in some contexts relies on user interaction. This means that responses might not arrive in the order in which the requests were made.

If a request for authentication is accepted, the sequence of Certificate, CertificateVerify, and Finished messages are sent by the responding peer. As with spontaneous authentication, these messages MUST form a contiguous sequence.

A request for authentication can be rejected by sending a Certificate message that contains an empty certificate_list field. The extensions field of this message MUST be empty.

A client MUST NOT request server authentication unless the server included client_auth_solicited in its "post_handshake_auth" extension. Similarly, a server MUST NOT request client authentication unless it included client_auth_solicited in its "post_handshake_auth" extension.

3. Post-Handshake Authentication TLS Extension

The "post_handshake_auth" TLS extension advertises support for post-handshake authentication.

```
      enum {
          client_auth_solicited(0),
          client_auth_spontaneous(1),
          server_auth_solicited(2),
          server_auth_spontaneous(3),
          (255)
      } AuthTypes;

      struct {
          AuthType auth_types<0..2^8-1>;
      } PostHandshakeAuth;
```

The extension data for the "post_handshake_auth" extension is PostHandshakeAuth. This includes one or more AuthType. Each AuthType value represents support for a given authentication flow:

client_auth_solicited: indicates support for client authentication solicited by a server request

client_auth_spontaneous: indicates support for spontaneous client authentication

server_auth_solicited: indicates support for server authentication solicited by a client request

server_auth_spontaneous: indicates support for spontaneous server authentication

The client includes a "post_handshake_auth" extension containing every type of authentication flow it supports in its ClientHello. The server replies with an EncryptedExtensions containing a "post_handshake_auth" extension containing a list of authentication types that it supports. The set of AuthTypes in the server's "post_handshake_auth" extension MUST be a subset of those sent by the client.

The "post_handshake_auth" extension MUST be omitted if the server does not support any mode of post-handshake authentication in common with the client.

If a server declares support for either client_auth_solicited, or client_auth_spontaneous, it MUST also include a "signature_algorithms" extension (see Section 4.2.2 of [I-D.ietf-tls-tls13]). This contains a list of the signature schemes that the server is able to use for client authentication, listed in descending order of preference.

This extension is not compatible with the raw public key extension [RFC7250]. The server MUST NOT select the raw public key extension if it uses this mechanism.

4. Post-Handshake Authentication Messages

The messages used for post-handshake authentication closely mirror those used to authenticate certificates in the standard TLS handshake.

4.1. Certificate Request

For solicited post-handshake authentication, the first message is used to define the characteristics required in the solicited certificate.

```
  opaque DistinguishedName<1..2^16-1>;

  struct {
      opaque certificate_extension_oid<1..2^8-1>;
      opaque certificate_extension_values<0..2^16-1>;
  } CertificateExtension;

  struct {
      opaque certificate_request_context<1..2^8-1>;
      select (Role) {
          case server:
              DistinguishedName certificate_authorities<0..2^16-1>;
              CertificateExtension certificate_extensions<0..2^16-1>;
          case client:
              HostName host_name<1..2^16-1>;
      }
  } CertificateRequest;
```

The certificate_request_context is an opaque string which identifies the certificate request and which will be echoed in the corresponding Certificate message. The certificate_request_context value MUST be unique for the connection. A client MUST set the most significant bit of the first octet of the certificate_request_context; a server MUST clear this bit.

For CertificateRequests sent from the server, the DistinguishedName and CertificateExtension fields are defined exactly as in the TLS 1.3 specification.

For CertificateRequests send from the client, a HostName containing the Server Name Indication (defined in [RFC6066]) used for selecting the certificate is included.

4.2. Certificate Message

The certificate message is used to transport the certificate. It mirrors the Certificate message in the TLS with the addition of some certificate-specific extensions.

```
opaque ASN1Cert<1..2^24-1>;

struct {
    opaque certificate_request_context<0..2^8-1>;
    ASN1Cert certificate_list<0..2^24-1>;
    Extension extensions<0..2^16-1>;
} Certificate;
```

certificate_request_context: If this message is in response to a CertificateRequest, the value of certificate_request_context in that message.

certificate_list: This is a sequence (chain) of certificates. The sender's end entity certificate MUST come first in the list. Each following certificate SHOULD directly certify one preceding it. Because certificate validation requires that trust anchors be distributed independently, a certificate that specifies a trust anchor MAY be omitted from the chain, provided that supported peers are known to possess any omitted certificates.

extensions: Valid extensions include OCSP Status extensions ([RFC6066] and [RFC6961]) and SignedCertificateTimestamps ([RFC6962]). Any extension presented in a Certificate message must only be presented if the associated ClientHello extension was presented in the initial handshake.

The certificate_request_context is an opaque string that identifies the certificate. The certificate_request_context value MUST be unique for the connection. If the certificate is used in response to a CertificateRequest, certificate_request_context includes the certificate_request_context value in the corresponding CertificateRequest. If the Certificate message part of spontaneous authentication, the certificate_request_context value is chosen by the sender. When spontaneous authentication is used, a client MUST set the most significant bit of the first octet of the certificate_request_context; a server MUST clear this bit.

Any certificates provided MUST be signed using a signature scheme found in the "signature_algorithms" extension provided by the peer in the initial handshake. The end entity certificate MUST allow the key to be used for signing (i.e., the digitalSignature bit MUST be set if the Key Usage extension is present) with a signature scheme indicated

in the "signature_algorithms" extension provided by the peer in the initial handshake.

4.3. CertificateVerify Message

The CertificateVerify message used in this document is defined in Section 4.3.2. of [I-D.ietf-tls-tls13].

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

The algorithm field specifies the signature algorithm used (see Section 4.2.2 of [I-D.ietf-tls-tls13]). The signature is a digital signature using that algorithm that covers the handshake context, the resumption context and a hash of the CertificateRequest and Certificate messages:

```
Hash(handshake_context) + resumption_context +
    Hash(CertificateRequest* + Certificate)
```

Note that the CertificateRequest message is omitted with spontaneous authentication.

The value of handshake_context is the entire transcript of the initial handshake, starting from the first ClientHello up to the final Finished message from the client. The value of resumption_context is defined in Section 4.4.1 of [I-D.ietf-tls-tls13].

The context string that is input to the digital signature is formed by taking the endpoint role and the authentication mode. The final value is the concatenation of the ASCII-encoded strings:

- "TLS 1.3, "
- either "client" if the client is authenticating, or "server" if the server is authenticating
- a single space " " (0x20)
- "spontaneous" if no request was made; "solicited" if the peer sent a CertificateRequest
- " CertificateVerify"

Thus, a client that is responding to a CertificateRequest will use the string "TLS 1.3, client solicited CertificateVerify" as the context string.

4.4. Finished Message

Finished is defined in Section 4.3.3 of [I-D.ietf-tls-tls13]. When included in post-handshake authentication it includes a MAC over the value:

```
Hash(Handshake Context) + resumption_context +
    Hash(CertificateRequest* + Certificate + CertificateVerify)
```

Note that the CertificateRequest message is omitted with spontaneous authentication.

The Finished message uses the current traffic secret (traffic_secret_N) as the MAC key; the hash function and HMAC function are the negotiated PRF hash function.

4.5. Forgetting certificates

Certificate identity should not be maintained across resumption. If a connection is resumed, additional certificate identities for both client and server certificates SHOULD be forgotten. Either the client or the server MAY choose to forget a certificate identity at any time.

Repeated requests for the same certificate should be expected. If multiple certificate requests are recieved that differ only in the certificate_request_context value, it is permitted to only answer the most recent request.

5. Security Considerations

TBD

6. Acknowledgements

Eric Rescorla and Andrei Popov were involved in helpful discussions around this draft.

7. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-13 (work in progress), May 2016.

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

[RFC6961] Pettersen, Y., "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension", RFC 6961, DOI 10.17487/RFC6961, June 2013, <http://www.rfc-editor.org/info/rfc6961>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <http://www.rfc-editor.org/info/rfc6962>.

[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, <http://www.rfc-editor.org/info/rfc7250>.

Authors' Addresses

Nick Sullivan
CloudFlare Inc.

Email: nick@cloudflare.com

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

Mike Bishop
Microsoft

Email: michael.bishop@microsoft.com