Network Working Group   M. Thomson
Internet-Draft   Mozilla
Intended status: Standards Track   R. Hamilton
Expires: April 28, 2017   Google
                                                        October 25, 2016

Using Transport Layer Security (TLS) to Secure QUIC

draft-thomson-quic-tls-01

Abstract

This document describes how Transport Layer Security (TLS) can be used to secure QUIC.

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 April 28, 2017.

Copyright Notice

Copyright (c) 2016 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 (http://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 include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

QUIC [I-D.hamilton-quic-transport-protocol] provides a multiplexed transport for HTTP [RFC7230] semantics that provides several key advantages over HTTP/1.1 [RFC7230] or HTTP/2 [RFC7540] over TCP [RFC0793].

This document describes how QUIC can be secured using Transport Layer Security (TLS) version 1.3 [I-D.ietf-tls-tls13]. TLS 1.3 provides critical latency improvements for connection establishment over previous versions. Absent packet loss, most new connections can be established and secured within a single round trip; on subsequent connections between the same client and server, the client can often send application data immediately, that is, zero round trip setup.

This document describes how the standardized TLS 1.3 can act a security component of QUIC. The same design could work for TLS 1.2, though few of the benefits QUIC provides would be realized due to the handshake latency in versions of TLS prior to 1.3.

1.1. Notational Conventions

The words "MUST", "MUST NOT", "SHOULD", and "MAY" are used in this document. It's not shouting; when they are capitalized, they have the special meaning defined in [RFC2119].

2. Protocol Overview

QUIC [I-D.hamilton-quic-transport-protocol] can be separated into several modules:

1. The basic frame envelope describes the common packet layout. This layer includes connection identification, version negotiation, and includes markers that allow the framing and public reset to be identified.

2. The public reset is an unprotected packet that allows an intermediary (an entity that is not part of the security context) to request the termination of a QUIC connection.

3. Version negotiation frames are used to agree on a common version of QUIC to use.

4. Framing comprises most of the QUIC protocol. Framing provides a number of different types of frame, each with a specific purpose. Framing supports frames for both congestion management and stream multiplexing. Framing additionally provides a liveness testing capability (the PING frame).

5. Encryption provides confidentiality and integrity protection for frames. All frames are protected based on keying material derived from the TLS connection running on stream 1. Prior to this, data is protected with the 0-RTT keys.

6. Multiplexed streams are the primary payload of QUIC. These provide reliable, in-order delivery of data and are used to carry the encryption handshake and transport parameters (stream 1), HTTP header fields (stream 3), and HTTP requests and responses.

Frames for managing multiplexing include those for creating and destroying streams as well as flow control and priority frames.

7. Congestion management includes packet acknowledgment and other signal required to ensure effective use of available link capacity.

8. A complete TLS connection is run on stream 1. This includes the entire TLS record layer. As the TLS connection reaches certain states, keying material is provided to the QUIC encryption layer for protecting the remainder of the QUIC traffic.

9. HTTP mapping provides an adaptation to HTTP that is based on HTTP/2.

The relative relationship of these components are pictorally represented in Figure 1.

```
+-----+------+
| TLS | HTTP |
+-----+------+------------+
|  Streams   | Congestion |
+------------+------------+
|         Frames          +--------+---------+
+   +---------------------+ Public | Version |
|   |     Encryption      | Reset  |  Nego.  |
+---+---------------------+--------+---------+
|                   Envelope                 |
+--------------------------------------------+
|                     UDP                    |
+--------------------------------------------+
```

Figure 1: QUIC Structure

This document defines the cryptographic parts of QUIC. This includes the handshake messages that are exchanged on stream 1, plus the record protection that is used to encrypt and authenticate all other frames.

2.1. Handshake Overview

TLS 1.3 provides two basic handshake modes of interest to QUIC:

- A full handshake in which the client is able to send application data after one round trip and the server immediately after receiving the first message from the client.

- A 0-RTT handshake in which the client uses information about the server to send immediately. This data can be replayed by an attacker so it MUST NOT carry a self-contained trigger for any non-idempotent action.

A simplified TLS 1.3 handshake with 0-RTT application data is shown in Figure 2, see [I-D.ietf-tls-tls13] for more options and details.

```
    Client                                             Server

    ClientHello
   (Finished)
   (0-RTT Application Data)
   (end_of_early_data)        -------->
                                                  ServerHello
                                         {EncryptedExtensions}
                                         {ServerConfiguration}
                                                 {Certificate}
                                           {CertificateVerify}
                                                    {Finished}
                             <--------      [Application Data]
   {Finished}                -------->

   [Application Data]        <------->      [Application Data]
```

Figure 2: TLS Handshake with 0-RTT

Two additional variations on this basic handshake exchange are relevant to this document:

- The server can respond to a ClientHello with a HelloRetryRequest, which adds an additional round trip prior to the basic exchange. This is needed if the server wishes to request a different key exchange key from the client. HelloRetryRequest is also used to verify that the client is correctly able to receive packets on the address it claims to have (see Section 6.3).

- A pre-shared key mode can be used for subsequent handshakes to avoid public key operations. This is the basis for 0-RTT data, even if the remainder of the connection is protected by a new Diffie-Hellman exchange.

3. TLS in Stream 1

QUIC completes its cryptographic handshake on stream 1, which means that the negotiation of keying material happens after the QUIC protocol has started. This simplifies the use of TLS since QUIC is able to ensure that the TLS handshake packets are delivered reliably and in order.

QUIC Stream 1 carries a complete TLS connection. This includes the TLS record layer in its entirety. QUIC provides for reliable and in-order delivery of the TLS handshake messages on this stream.

Prior to the completion of the TLS handshake, QUIC frames can be exchanged. However, these frames are not authenticated or confidentiality protected. Section 5 covers some of the implications of this design and limitations on QUIC operation during this phase.

Once complete, QUIC frames are protected using QUIC record protection, see Section 4.

3.1. Handshake and Setup Sequence

The integration of QUIC with a TLS handshake is shown in more detail in Figure 3. QUIC "STREAM" frames on stream 1 carry the TLS handshake. QUIC is responsible for ensuring that the handshake packets are re-sent in case of loss and that they can be ordered correctly.

```
       Client                                             Server

   @A QUIC STREAM Frame(s) <1>:
        ClientHello
          + QUIC Setup Parameters
                               -------->
                            0-RTT Key -> @B

   @B QUIC STREAM Frame(s) <1>:
        (Finished)
      Replayable QUIC Frames <any stream>
                               -------->

                                         QUIC STREAM Frame <1>: @B/A
                                                  ServerHello
                                         {Handshake Messages}
                               <--------
                           1-RTT Key -> @C

                                                    QUIC Frames @C
                               <--------
   @B QUIC STREAM Frame(s) <1>:
        (end_of_early_data <1>)
        {Finished}
                               -------->

   @C QUIC Frames              <------->            QUIC Frames @C
```

Figure 3: QUIC over TLS Handshake

In Figure 3, symbols mean:

- "<" and ">" enclose stream numbers.
- "@" indicates the key phase that is currently used for protecting QUIC packets.
- "(" and ")" enclose messages that are protected with TLS 0-RTT handshake or application keys.
- "{" and "}" enclose messages that are protected by the TLS Handshake keys.

If 0-RTT is not possible, then the client does not send frames protected by the 0-RTT key (@B). The only key transition on the client is from cleartext (@A) to 1-RTT protection (@C).

If 0-RTT data is not accepted by the server, then the server sends its handshake messages without protection (@A). The client still transitions from @A to @B, but it can stop sending 0-RTT data and progress immediately to 1-RTT data when it receives a cleartext ServerHello.

4. QUIC Record Protection

QUIC provides a record protection layer that is responsible for authenticated encryption of packets. The record protection layer uses keys provided by the TLS connection and authenticated encryption to provide confidentiality and integrity protection for the content of packets.

Different keys are used for QUIC and TLS record protection. Having separate QUIC and TLS record protection means that TLS records can be protected by two different keys. This redundancy is maintained for the sake of simplicity.

4.1. Key Phases

The transition to use of a new QUIC key occurs immediately after sending the TLS handshake messages that produced the key transition. Every time that a new set of keys is used for protecting outbound messages, the KEY_PHASE bit in the public flags is toggled. The KEY_PHASE bit on unencrypted messages is 0.

The KEY_PHASE bit on the public flags is the most significant bit (0x80).

The KEY_PHASE bit allows a recipient to detect a change in keying material without needing to receive the message that triggers the change. This avoids head-of-line blocking around transitions between keys without relying on trial decryption.

The following transitions are defined:

- The client transitions to using 0-RTT keys after sending the ClientHello. This causes the KEY_PHASE bit on packets sent by the client to be set to 1.

- The server transitions to using 0-RTT keys before sending the ServerHello, but only if the early data from the client is accepted. This transition causes the KEY_PHASE bit on packets sent by the server to be set to 1. If the server rejects 0-RTT data, the server's handshake messages are sent without QUIC-level record protection with a KEY_PHASE of 0. TLS handshake messages will still be protected by TLS record protection based on the TLS handshake traffic keys.

- The server transitions to using 1-RTT keys after sending its Finished message. This causes the KEY_PHASE bit to be set to 0 if early data was accepted, and 1 if the server rejected early data.

- The client transitions to 1-RTT keys after sending its Finished message. Subsequent messages from the client will then have a KEY_PHASE of 0 if 0-RTT data was sent, and 1 otherwise.

- Both peers start sending messages protected by a new key immediately after sending a TLS KeyUpdate message. The value of the KEY_PHASE bit is changed each time.

At each point, both keying material (see Section 4.2) and the AEAD function used by TLS is interchanged with the values that are currently in use for protecting outbound packets. Once a change of keys has been made, packets with higher sequence numbers MUST use the new keying material until a newer set of keys (and AEAD) are used. The exception to this is that retransmissions of TLS handshake packets MUST use the keys that they were originally protected with.

Once a packet protected by a new key has been received, a recipient SHOULD retain the previous keys for a short period. Retaining old keys allows the recipient to decode reordered packets around a change in keys. Keys SHOULD be discarded when an endpoints has received all packets with sequence numbers lower than the lowest sequence number used for the new key, or when it determines that reordering of those packets is unlikely. 0-RTT keys SHOULD be retained until the handshake is complete.

The KEY_PHASE bit does not directly indicate which keys are in use. Depending on whether 0-RTT data was sent and accepted, packets protected with keys derived from the same secret might be marked with different KEY_PHASE values.

4.1.1. Retransmission of TLS Handshake Messages

TLS handshake messages need to be retransmitted with the same level of cryptographic protection that was originally used to protect them. Newer keys cannot be used to protect QUIC packets that carry TLS messages.

A client would be unable to decrypt retransmissions of a server's handshake messages that are protected using the 1-RTT keys, since the calculation of the application data keys depends on the contents of the handshake messages.

This restriction means the creation of an exception to the requirement to always use new keys for sending once they are available. A server MUST mark the retransmitted handshake messages with the same KEY_PHASE as the original messages to allow a recipient to distinguish the messages.

4.1.2. Key Update

Once the TLS handshake is complete, the KEY_PHASE bit allows for the processing of messages without having to receive the TLS KeyUpdate message that triggers the key update. This allows endpoints to start using updated keys immediately without the concern that a lost KeyUpdate will cause their messages to be indecipherable to their peer..

An endpoint MUST NOT initiate more than one key update at a time. A new key update cannot be sent until the endpoint has received a matching KeyUpdate message from its peer; or, if the endpoint did not initiate the original key update, it has received an acknowledgment of its own KeyUpdate.

This ensures that there are at most two keys to distinguish between at any one time, for which the KEY_PHASE bit is sufficient.

```
   Initiating Peer                    Responding Peer

@M KeyUpdate
                    New Keys -> @N
@N QUIC Frames
                      -------->
                                            KeyUpdate @N
                      <--------
  -- Initiating Peer can initiate another KeyUpdate --
 @N Acknowledgment
                      -------->
  -- Responding Peer can initiate another KeyUpdate --
```

Figure 4: Key Update

As shown in Figure 3 and Figure 4, there is never a situation where there are more than two different sets of keying material that might be received by a peer.

A server cannot initiate a key update until it has received the client's Finished message. Otherwise, packets protected by the updated keys could be confused for retransmissions of handshake messages. A client cannot initiate a key update until it has received an acknowledgment that its Finished message has been received.

Note: This models the key changes in the handshake as a key update initiated by the server, with the Finished message in the place of KeyUpdate.

4.2. QUIC Key Expansion

The following table shows QUIC keys, when they are generated and the TLS secret from which they are derived:

Key	TLS Secret	Phase
0-RTT	early_traffic_secret	"QUIC 0-RTT key expansion"
1-RTT	traffic_secret_N	"QUIC 1-RTT key expansion"

0-RTT keys are those keys that are used in resumed connections prior to the completion of the TLS handshake. Data sent using 0-RTT keys might be replayed and so has some restrictions on its use, see Section 5.2. 0-RTT keys are used after sending or receiving a ClientHello.

1-RTT keys are used after the TLS handshake completes. There are potentially multiple sets of 1-RTT keys; new 1-RTT keys are created by sending a TLS KeyUpdate message. 1-RTT keys are used after sending a Finished or KeyUpdate message.

The complete key expansion uses the same process for key expansion as defined in Section 7.3 of [I-D.ietf-tls-tls13]. For example, the Client Write Key for the data sent immediately after sending the TLS Finished message is:

```
label = "QUIC 1-RTT key expansion, client write key"
client_write = HKDF-Expand-Label(traffic_secret_0, label,
                                 "", key_length)
```

This results in a label input to HKDF that includes a two-octet length field, the string "TLS 1.3, QUIC 1-RTT key expansion, client write key" and a zero octet.

The QUIC record protection initially starts without keying material. When the TLS state machine produces the corresponding secret, new keys are generated from the TLS connection and used to protect the QUIC record protection.

The Authentication Encryption with Associated Data (AEAD) [RFC5116] function used is the same one that is negotiated for use with the TLS connection. For example, if TLS is using the TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256, the AEAD_AES_128_GCM function is used.

4.3. QUIC AEAD application

Regular QUIC packets are protected by an AEAD [RFC5116]. Version negotiation and public reset packets are not protected.

Once TLS has provided a key, the contents of regular QUIC packets immediately after any TLS messages have been sent are protected by the AEAD selected by TLS.

The key, K, for the AEAD is either the Client Write Key or the Server Write Key, derived as defined in Section 4.2.

The nonce, N, for the AEAD is formed by combining either the Client Write IV or Server Write IV with the sequence numbers. The 48 bits of the reconstructed QUIC sequence number (see Section 4.4) in network byte order is left-padded with zeros to the N_MAX parameter of the AEAD (see Section 4 of [RFC5116]). The exclusive OR of the padded sequence number and the IV forms the AEAD nonce.

The associated data, A, for the AEAD is an empty sequence.

The input plaintext, P, for the AEAD is the contents of the QUIC frame following the packet number, as described in [I-D.hamilton-quic-transport-protocol]

The output ciphertext, C, of the AEAD is transmitted in place of P.

Prior to TLS providing keys, no record protection is performed and the plaintext, P, is transmitted unmodified.

Note: QUIC defined a null-encryption that had an additional, hash-based checksum for cleartext packets. This might be added here, but it is more complex.

4.4. Sequence Number Reconstruction

Each peer maintains a 48-bit sequence number that is incremented with every packet that is sent, including retransmissions. The least significant 8-, 16-, 32-, or 48-bits of this number is encoded in the QUIC sequence number field in every packet.

A receiver maintains the same values, but recovers values based on the packets it receives. This is based on the sequence number of packets that it has received. A simple scheme predicts the receive sequence number of an incoming packet by incrementing the sequence number of the most recent packet to be successfully decrypted by one and expecting the sequence number to be within a range centered on that value.

A more sophisticated algorithm can almost double the search space by checking backwards from the most recent sequence for a received (or abandoned) packet. If a packet was received, then the packet contains a sequence number that is greater than the most recent sequence number. If no such packet was found, the number is assumed to be in the smaller window centered on the next sequence number, as in the simpler scheme.

Note: QUIC has a single, contiguous sequence number space. In comparison, TLS restarts its sequence number each time that record protection keys are changed. The sequence number restart in TLS ensures that a compromise of the current traffic keys does not allow an attacker to truncate the data that is sent after a key update by sending additional packets under the old key (causing new packets to be discarded). QUIC does not assume a reliable transport and is therefore required to handle attacks where packets are dropped in other ways. TLS maintains a separate sequence number that is used for record protection on the connection that is hosted on stream 1. This sequence number is reset according to the rules in the TLS protocol.

5. Pre-handshake QUIC Messages

Implementations MUST NOT exchange data on any stream other than stream 1 prior to the completion of the TLS handshake. However, QUIC requires the use of several types of frame for managing loss detection and recovery. In addition, it might be useful to use the data acquired during the exchange of unauthenticated messages for congestion management.

This section generally only applies to TLS handshake messages from both peers and acknowledgments of the packets carrying those messages. In many cases, the need for servers to provide acknowledgments is minimal, since the messages that clients send are small and implicitly acknowledged by the server's responses.

The actions that a peer takes as a result of receiving an unauthenticated packet needs to be limited. In particular, state established by these packets cannot be retained once record protection commences.

There are several approaches possible for dealing with unauthenticated packets prior to handshake completion:

- discard and ignore them
- use them, but reset any state that is established once the handshake completes
- use them and authenticate them afterwards; failing the handshake if they can't be authenticated
- save them and use them when they can be properly authenticated
- treat them as a fatal error

Different strategies are appropriate for different types of data. This document proposes that all strategies are possible depending on the type of message.

- Transport parameters and options are made usable and authenticated as part of the TLS handshake (see Section 6.2).
- Most unprotected messages are treated as fatal errors when received except for the small number necessary to permit the handshake to complete (see Section 5.1).
- Protected packets can either be discarded or saved and later used (see Section 5.3).

5.1. Unprotected Frames Prior to Handshake Completion

This section describes the handling of messages that are sent and received prior to the completion of the TLS handshake.

Sending and receiving unprotected messages is hazardous. Unless expressly permitted, receipt of an unprotected message of any kind MUST be treated as a fatal error.

5.1.1. STREAM Frames

"STREAM" frames for stream 1 are permitted. These carry the TLS handshake messages.

Receiving unprotected "STREAM" frames for other streams MUST be treated as a fatal error.

5.1.2. ACK Frames

"ACK" frames are permitted prior to the handshake being complete. Information learned from "ACK" frames cannot be entirely relied upon, since an attacker is able to inject these packets. Timing and packet retransmission information from "ACK" frames is critical to the functioning of the protocol, but these frames might be spoofed or altered.

Endpoints MUST NOT use an unprotected "ACK" frame to acknowledge data that was protected by 0-RTT or 1-RTT keys. An endpoint MUST ignore an unprotected "ACK" frame if it claims to acknowledge data that was protected data. Such an acknowledgement can only serve as a denial of service, since an endpoint that can read protected data is always permitted to send protected data.

An endpoint SHOULD use data from unprotected or 0-RTT-protected "ACK" frames only during the initial handshake and while they have insufficient information from 1-RTT-protected "ACK" frames. Once sufficient information has been obtained from protected messages, information obtained from less reliable sources can be discarded.

5.1.3. WINDOW_UPDATE Frames

"WINDOW_UPDATE" frames MUST NOT be sent unprotected.

Though data is exchanged on stream 1, the initial flow control window is is sufficiently large to allow the TLS handshake to complete. This limits the maximum size of the TLS handshake and would prevent a server or client from using an abnormally large certificate chain.

Stream 1 is exempt from the connection-level flow control window.

5.1.4. Denial of Service with Unprotected Packets

Accepting unprotected - specifically unauthenticated - packets presents a denial of service risk to endpoints. An attacker that is able to inject unprotected packets can cause a recipient to drop even protected packets with a matching sequence number. The spurious packet shadows the genuine packet, causing the genuine packet to be ignored as redundant.

Once the TLS handshake is complete, both peers MUST ignore unprotected packets. The handshake is complete when the server receives a client's Finished message and when a client receives an acknowledgement that their Finished message was received. From that point onward, unprotected messages can be safely dropped. Note that the client could retransmit its Finished message to the server, so the server cannot reject such a message.

Since only TLS handshake packets and acknowledgments are sent in the clear, an attacker is able to force implementations to rely on retransmission for packets that are lost or shadowed. Thus, an attacker that intends to deny service to an endpoint has to drop or shadow protected packets in order to ensure that their victim continues to accept unprotected packets. The ability to shadow packets means that an attacker does not need to be on path.

ISSUE: This would not be an issue if QUIC had a randomized starting sequence number. If we choose to randomize, we fix this problem and reduce the denial of service exposure to on-path attackers. The only possible problem is in authenticating the initial value, so that peers can be sure that they haven't missed an initial message.

In addition to denying endpoints messages, an attacker to generate packets that cause no state change in a recipient. See Section 7.2 for a discussion of these risks.

To avoid receiving TLS packets that contain no useful data, a TLS implementation MUST reject empty TLS handshake records and any record that is not permitted by the TLS state machine. Any TLS application data or alerts - other than a single end_of_early_data at the appropriate time - that is received prior to the end of the handshake MUST be treated as a fatal error.

5.2. Use of 0-RTT Keys

If 0-RTT keys are available, the lack of replay protection means that restrictions on their use are necessary to avoid replay attacks on the protocol.

A client MUST only use 0-RTT keys to protect data that is idempotent. A client MAY wish to apply additional restrictions on what data it sends prior to the completion of the TLS handshake. A client otherwise treats 0-RTT keys as equivalent to 1-RTT keys.

A client that receives an indication that its 0-RTT data has been accepted by a server can send 0-RTT data until it receives all of the server's handshake messages. A client SHOULD stop sending 0-RTT data if it receives an indication that 0-RTT data has been rejected. In addition to a ServerHello without an early_data extension, an unprotected handshake message with a KEY_PHASE bit set to 0 indicates that 0-RTT data has been rejected.

A client SHOULD send its end_of_early_data alert only after it has received all of the server's handshake messages. Alternatively phrased, a client is encouraged to use 0-RTT keys until 1-RTT keys become available. This prevents stalling of the connection and allows the client to send continuously.

A server MUST NOT use 0-RTT keys to protect anything other than TLS handshake messages. Servers therefore treat packets protected with 0-RTT keys as equivalent to unprotected packets in determining what is permissible to send. A server protects handshake messages using the 0-RTT key if it decides to accept a 0-RTT key. A server MUST still include the early_data extension in its ServerHello message.

This restriction prevents a server from responding to a request using frames protected by the 0-RTT keys. This ensures that all application data from the server are always protected with keys that have forward secrecy. However, this results in head-of-line blocking at the client because server responses cannot be decrypted until all the server's handshake messages are received by the client.

5.3. Protected Frames Prior to Handshake Completion

Due to reordering and loss, protected packets might be received by an endpoint before the final handshake messages are received. If these can be decrypted successfully, such packets MAY be stored and used once the handshake is complete.

Unless expressly permitted below, encrypted packets MUST NOT be used prior to completing the TLS handshake, in particular the receipt of a valid Finished message and any authentication of the peer. If packets are processed prior to completion of the handshake, an attacker might use the willingness of an implementation to use these packets to mount attacks.

TLS handshake messages are covered by record protection during the handshake, once key agreement has completed. This means that protected messages need to be decrypted to determine if they are TLS handshake messages or not. Similarly, "ACK" and "WINDOW_UPDATE" frames might be needed to successfully complete the TLS handshake.

Any timestamps present in "ACK" frames MUST be ignored rather than causing a fatal error. Timestamps on protected frames MAY be saved and used once the TLS handshake completes successfully.

An endpoint MAY save the last protected "WINDOW_UPDATE" frame it receives for each stream and apply the values once the TLS handshake completes. Failing to do this might result in temporary stalling of affected streams.

6. QUIC-Specific Additions to the TLS Handshake

QUIC uses the TLS handshake for more than just negotiation of cryptographic parameters. The TLS handshake validates protocol version selection, provides preliminary values for QUIC transport parameters, and allows a server to perform return routeability checks on clients.

6.1. Protocol and Version Negotiation

The QUIC version negotiation mechanism is used to negotiate the version of QUIC that is used prior to the completion of the handshake. However, this packet is not authenticated, enabling an active attacker to force a version downgrade.

To ensure that a QUIC version downgrade is not forced by an attacker, version information is copied into the TLS handshake, which provides integrity protection for the QUIC negotiation. This does not prevent version downgrade during the handshake, though it means that such a downgrade causes a handshake failure.

Protocols that use the QUIC transport MUST use Application Layer Protocol Negotiation (ALPN) [RFC7301]. The ALPN identifier for the protocol MUST be specific to the QUIC version that it operates over. When constructing a ClientHello, clients MUST include a list of all the ALPN identifiers that they support, regardless of whether the QUIC version that they have currently selected supports that protocol.

Servers SHOULD select an application protocol based solely on the information in the ClientHello, not using the QUIC version that the client has selected. If the protocol that is selected is not supported with the QUIC version that is in use, the server MUST either send a QUIC version negotiation packet if this is possible, or fail the connection otherwise.

6.2. QUIC Extension

QUIC defines an extension for use with TLS. That extension defines transport-related parameters. This provides integrity protection for these values. Including these in the TLS handshake also make the values that a client sets available to a server one-round trip earlier than parameters that are carried in QUIC frames. This document does not define that extension.

6.3. Source Address Validation

QUIC implementations describe a source address token. This is an opaque blob that a server might provide to clients when they first use a given source address. The client returns this token in subsequent messages as a return routeability check. That is, the client returns this token to prove that it is able to receive packets at the source address that it claims. This prevents the server from being used in packet reflection attacks (see Section 7.1).

A source address token is opaque and consumed only by the server. Therefore it can be included in the TLS 1.3 pre-shared key identifier for 0-RTT handshakes. Servers that use 0-RTT are advised to provide new pre-shared key identifiers after every handshake to avoid linkability of connections by passive observers. Clients MUST use a new pre-shared key identifier for every connection that they initiate; if no pre-shared key identifier is available, then resumption is not possible.

A server that is under load might include a source address token in the cookie extension of a HelloRetryRequest. (Note: the current version of TLS 1.3 does not include the ability to include a cookie in HelloRetryRequest.)

6.4. Priming 0-RTT

QUIC uses TLS without modification. Therefore, it is possible to use a pre-shared key that was obtained in a TLS connection over TCP to enable 0-RTT in QUIC. Similarly, QUIC can provide a pre-shared key that can be used to enable 0-RTT in TCP.

All the restrictions on the use of 0-RTT apply, and the certificate MUST be considered valid for both connections, which will use different protocol stacks and could use different port numbers. For instance, HTTP/1.1 and HTTP/2 operate over TLS and TCP, whereas QUIC operates over UDP.

Source address validation is not completely portable between different protocol stacks. Even if the source IP address remains constant, the port number is likely to be different. Packet reflection attacks are still possible in this situation, though the set of hosts that can initiate these attacks is greatly reduced. A server might choose to avoid source address validation for such a connection, or allow an increase to the amount of data that it sends toward the client without source validation.

7. Security Considerations

There are likely to be some real clangers here eventually, but the current set of issues is well captured in the relevant sections of the main text.

Never assume that because it isn't in the security considerations section it doesn't affect security. Most of this document does.

7.1. Packet Reflection Attack Mitigation

A small ClientHello that results in a large block of handshake messages from a server can be used in packet reflection attacks to amplify the traffic generated by an attacker.

Certificate caching [RFC7924] can reduce the size of the server's handshake messages significantly.

A client SHOULD also pad [RFC7685] its ClientHello to at least 1024 octets (TODO: tune this value). A server is less likely to generate a packet reflection attack if the data it sends is a small multiple of the data it receives. A server SHOULD use a HelloRetryRequest if the size of the handshake messages it sends is likely to exceed the size of the ClientHello.

7.2. Peer Denial of Service

QUIC, TLS and HTTP/2 all contain a messages that have legitimate uses in some contexts, but that can be abused to cause a peer to expend processing resources without having any observable impact on the state of the connection. If processing is disproportionately large in comparison to the observable effects on bandwidth or state, then this could allow a malicious peer to exhaust processing capacity without consequence.

QUIC prohibits the sending of empty "STREAM" frames unless they are marked with the FIN bit. This prevents "STREAM" frames from being sent that only waste effort.

TLS records SHOULD always contain at least one octet of a handshake messages or alert. Records containing only padding are permitted during the handshake, but an excessive number might be used to generate unnecessary work. Once the TLS handshake is complete, endpoints SHOULD NOT send TLS application data records unless it is to hide the length of QUIC records. QUIC packet protection does not include any allowance for padding; padded TLS application data records can be used to mask the length of QUIC frames.

While there are legitimate uses for some redundant packets, implementations SHOULD track redundant packets and treat excessive volumes of any non-productive packets as indicative of an attack.

8. IANA Considerations

This document has no IANA actions. Yet.

9. References

9.1. Normative References

[I-D.hamilton-quic-transport-protocol]
Hamilton, R., Iyengar, J., Swett, I., and A. Wilk, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-hamilton-quic-transport-protocol-00 (work in progress), July 2016.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-17 (work in progress), October 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>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <http://www.rfc-editor.org/info/rfc5116>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

9.2. Informative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[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>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <http://www.rfc-editor.org/info/rfc7540>.

[RFC7685] Langley, A., "A Transport Layer Security (TLS) ClientHello Padding Extension", RFC 7685, DOI 10.17487/RFC7685, October 2015, <http://www.rfc-editor.org/info/rfc7685>.

[RFC7924] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", RFC 7924, DOI 10.17487/RFC7924, July 2016, <http://www.rfc-editor.org/info/rfc7924>.

Appendix A. Acknowledgments

Christian Huitema's knowledge of QUIC is far better than my own. This would be even more inaccurate and useless if not for his assistance. This document has variously benefited from a long series of discussions with Jana Iyengar, Adam Langley, Roberto Peon, Eric Rescorla, Ian Swett, and likely many others who are merely forgotten by a faulty meat computer.

Authors' Addresses

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

Ryan Hamilton
Google

Email: rch@google.com