QUIC
Internet-Draft
Intended status: Experimental
Expires: June 17, 2019

K. Oku
Fastly
C. Huitema
Private Octopus Inc.
December 14, 2018

Authenticated Handshake for QUIC
draft-kazuho-quic-authenticated-handshake-00

Abstract

This document explains a variant of QUIC protocol version 1 that uses the ESNI Keys to authenticate the Initial packets thereby making the entire handshake tamper-proof.

Table of Contents

1. Introduction

As defined in Secure Using TLS to Secure QUIC [QUIC-TLS], QUIC version 1 [QUIC-TRANSPORT] protects the payload of every QUIC packet using AEAD making the protocol injection- and tamper-proof, with the exception being the Initial packets. Initial packets are merely obfuscated because there is no shared secret between the endpoints when they start sending the Initial packets against each other.

However, when Encrypted Server Name Indication for TLS 1.3 [TLS-ESNI] is used, a shared secret between the endpoints can be used for authentication from the very first packet of the connection.

This document defines a Packet Protection method for Initial packets that incorporates the ESNI shared secret, so that spoofed Initial packets will be detected and droped.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",

"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Differences from QUIC version 1

The document describes the changes from QUIC version 1.

Implementations MUST conform to the specifications of QUIC version 1 unless a different behavior is defined in this document.

2.1. Protocol Version Number

The long header packets exchanged using this specification carry the QUIC version number of 0xXXXXXXXX (TBD).

2.2. The "QUIC-ESNI" TLS Extension

The QUIC-ESNI TLS Extension indicates the versions of the QUIC protocol that the server supports. The values in the extension SHOULD be identical to what would be included in the Version Negotiation packet.

```
struct {
    uint32 supported_versions<4..2^16-4>;
} QUIC_ESNI;
```

A server willing to accept QUIC connections using this specification MUST publish ESNI Resource Records that contain the QUIC_ESNI extension including the QUIC version number 0xXXXXXXXX.

A client MUST NOT initiate a connection establishment attempt specified in this document unless it sees a compatible version number in the QUIC_ESNI extension of the ESNI Resource Record advertised by the server.

2.3. Initial Packet Protection

Initial packets are encrypted and authenticated differently from QUIC version 1.

AES [AES] in counter (CTR) mode is used for encrypting the payload. The key and iv being used are identical to that of QUIC version 1.

HMAC [RFC2104] is used for authenticating the header. The message being authenticated is the concatenation of the packet header without Header Protection and the payload in cleartext. The underlying hash function being used is the one selected for encrypting the Encrypted SNI extension. The HMAC key is calculated using the following formula:


```
hmac_key = HKDF-Expand-Label(Zx, "quic initial auth", Hash(ESNIContents),
                             digest_size)
```

The first sixteen (16) octets of the HMAC output replaces the authentication tag of QUIC version 1.

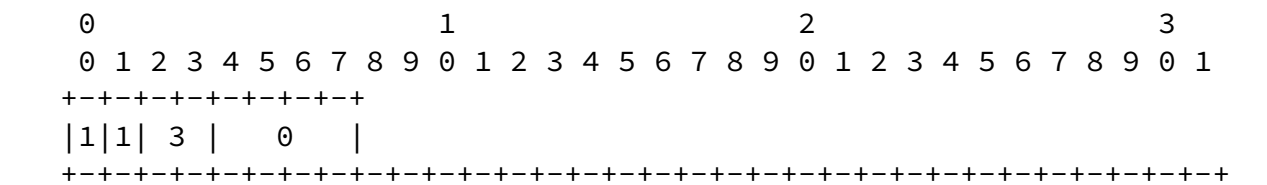
Other types of packets are protected using the Packet Protection method defined in QUIC version 1.

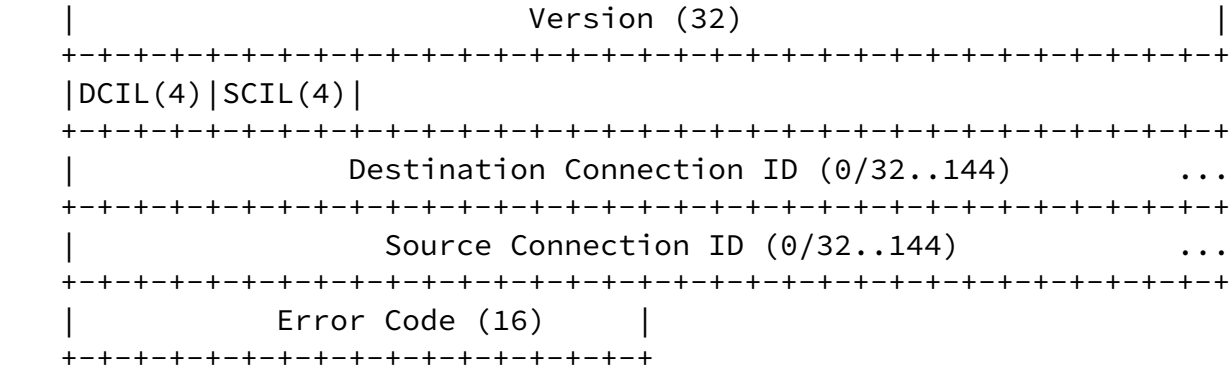
2.4. Version Negotiation Packet

A client MUST ignore Version Negotiation packets. When the client gives up of establishing a connection, it MAY report the failure differently based on the receipt of (or lack of) Version Negotiation packets.

2.5. Connection Close Packet

A Connection Close packet shares a long packet header with a type value of 0x3 with the Retry packet. The two types of packets are identified by the lower 4-bits of the first octet. The packet is a Connection Close packet if all the bits are set to zero. Otherwise, the packet is a Retry packet.





(: #connection-close-format title="Connection Close Packet")

A Connection Close packet is sent by a server when a connection error occurs prior to deriving the HMAC key. In all other conditions, connection close MUST be signalled using the CONNECTION_CLOSE frame.

A client that receives a Connection Close packet before an Initial packet SHOULD retain the error code, and continue the connection

establishment attempt as if it did not see the packet. When the attempt times out, it MAY assume that the error code was a legitimate value sent by the server. A client MAY ignore Connection Close packets.

2.6. Retry Packet

A client SHOULD send an Initial packet in response to each Retry packet it receives. Payload of the CRYPTO frame contained in the resent Initial packets MUST be identical to that of the Initial packet that triggered the retry. When the client does not receive a valid Initial packet after a handshake timeout, it SHOULD send at least one Initial packet containing one of the tokens that it has received. Unless the packet gets lost, the retransmission would trigger the server to send either a valid Initial packet or a Retry packet.

To a server, the behavior of a client under attack would look like it is aggressively retransmitting Initial packets, some of them containing invalid tokens.

Therefore, a server MUST NOT terminate the connection when it

receives an Initial packet that contains an invalid token. Instead, it SHOULD either process the packet as if it did not contain a token, or send a Retry.

A client MUST ignore Retry packets received anterior to an Initial packet that successfully authenticates.

3. Considerations

3.1. Using GCM to Authenticate Initial Packets

An alternative approach to using the combination of AES-CTR and HMAC is to continue using AES-GCM. In such approach, the additional authenticated data (AAD) will incorporate the ESNI shared secret to detect spoofed or broken packets.

A server that receives an Initial packet for a new connection will at first decrypt the payload using AES-CTR, derive ESNI shared secret from the Hello message being contained, then use that to verify the GCM tag.

The benefit of the approach is that we will have less divergence from QUIC version 1. The downside is that the authentication algorithm would be hard-coded to GCM, and that some AEAD APIs might not provide an interface to handle input in this particular way.

We can also consider adding a small checksum to the Initial packets so that the server can determine if the packet is corrupt. The downside is that the endpoints would be required to calculate the checksum for Initial packets that carry server's messages and ACKs as well, even though the correctness of the packet can be verified using the ordinary procedure of AEAD.

3.2. Use of Different QUIC Version Number

For this specification, use of a different QUIC version number is not expected to have negative impact on user-experience by raising the chance of version negotiation, because version negotiation finishes before the client sends it's first packet.

Use of Encrypted SNI will stick out more, because it can be

identified by observing a different version number in the long header packet rather than by decrypting the Initial packet to see if the Encrypted SNI extension is in use.

The subsections below discuss alternative approaches that do not change the version number of QUIC.

3.2.1. Trial Decryption

It is possible to use the proposed Packet Protection method without changing the version number. The difference from the recommended method is that the server would be required to do "trial decryption."

However, it is not as bad as it sounds, because authentication failure in AES-GCM decryption is typically reported after the ciphertext is decrypted.

When accepting a new connection, a QUIC server can at first decrypt the Initial packet using AES-GCM. The packet is a ordinary QUIC version 1 packet if it is successfully authenticated. Otherwise, the server will feed the decrypted payload (which would be available anyways) assuming that it contains a ClientHello message, and if the TLS stack successfully processes the message returning the handshake keys and the ESNI shared key, verify the HMAC to see if the packet authenticates. If it does, the server creates a new connection context and responds with an Initial packet.

3.2.2. Rekeying at the Server's First Flight

Another approach is to use the Packet Protection method of QUIC version 1 for client's first flight, while using the proposed method for all other Initial packets.

The benefit of this approach is that trial decryption can be avoided.

The downside is that a man-on-the-side attacker can stitch the Encrypted SNI extension that the client has sent with anything it wants to construct a spoofed packet, then race it to the server.

The server would be required to consider Initial packets containing non-identical ClientHello messages as belonging to different

connection establishment attempts.

The design will also have negative performance impact on connections with high latency. This is because QUIC expects clients to retransmit the Initial packets when the latency is above 250 milliseconds. However, the requirement that the server rekeys the Initial secret when receiving the first Initial packet means that the retransmitted Initial packets would become undecryptable and therefore be deemed lost by the client, reducing the client's congestion window size.

3.3. No Support for Split Mode

Under the design discussed in this document, it is impossible to use an unmodified QUIC server as a backend server in "Split Mode" ([TLS-ESNI]; section 3) due to the following two reasons:

- Access to initial_auth_secret is required for generating and validating Initial packets. However, the backend server, not knowing the ESNI private key, cannot calculate the secret.

- The client-facing server cannot continue forwarding packets to the correct destination when there is a change in Connection ID mid-connection.

To address the issues, we might consider specifying a protocol that will be used between the client-facing server and the backend server for communicating the initial_auth_secret and the spare Connection IDs. Note that such protocol can be lightweight, assuming the communication between the two servers will be over a virtual private network. Such assumption can be made because the backend server cannot operate QUIC without access to the source address-port tuple of the packets that the client has sent.

4. Security Considerations

TBD

5. IANA Considerations

TBD

6. Acknowledgements

TBD

7. Normative References

[AES] "Advanced encryption standard (AES)", National Institute of Standards and Technology report, DOI 10.6028/nist.fips.197, November 2001.

[QUIC-TLS] Thomson, M., Ed. and S. Turner, Ed., "Using Transport Layer Security (TLS) to Secure QUIC", draft-ietf-quic-tls-16 (work in progress), October 2018.

[QUIC-TRANSPORT] Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-16 (work in progress), October 2018.

[RFC2104] 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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[TLS-ESNI] Rescorla, E., Oku, K., Sullivan, N., and C. Wood, "Encrypted Server Name Indication for TLS 1.3", draft-ietf-tls-esni-02 (work in progress), October 2018.

Authors' Addresses

Kazuho Oku
Fastly

Email: kazuhooku@gmail.com

Christian Huitema
Private Octopus Inc.

Email: huitema@huitema.net