QUIC   K. Oku
Internet-Draft   Fastly
Intended status: Experimental   C. Huitema
Expires: January 6, 2020   Private Octopus Inc.
                                                           July 05, 2019

Authenticated Handshake for QUIC
draft-kazuho-quic-authenticated-handshake-01

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.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction 2
   1.1. Notational Conventions 3
2. Differences from QUIC version 1 3
   2.1. Protocol Version Number 3
   2.2. The "QUIC-AH" TLS Extension 3
   2.3. Initial Packet 4
      2.3.1. Mapping to Connections 4
      2.3.2. Protection 4
      2.3.3. Destination Connection ID 4
   2.4. Version Negotiation Packet 5
   2.5. Connection Close Packet 5
   2.6. Retry Packet 6
3. Considerations 6
   3.1. Using GCM to Authenticate Initial Packets 6
   3.2. Split Mode 7
4. Security Considerations 7
   4.1. Resisting the duplicate context attack 8
   4.2. Resisting Address Substitution Attacks 8
5. IANA Considerations 9
6. Normative References 9
Appendix A. Acknowledgements 10
Appendix B. Change Log 10
   B.1. Since draft-kazuho-quic-authenticated-handshake-00 . . . 10
Authors' Addresses 10

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-AH" TLS Extension

The QUIC-AH TLS Extension indicates the versions of QUIC supported by the server that have the authenticated handshake flavors, along with the versions being exposed on the wire for each of those versions.

```
struct {
    uint32 base_version;
    uint32 wire_versions<4..2^16-4>;
} SupportedVersion;

struct {
    SupportedVersion supported_versions<8..2^16-4>;
} QUIC_AH;
```

This specification defines a variant of QUIC version 1. Therefore, a ESNI Resource Records being published for a server providing support for this specification MUST include a QUIC_AH extension that contains a SupportedVersion structure with the "base_version" set to 1.

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

The "wire_versions" field indicates the version numbers to be contained in the long header packets, for each of the base versions that the server supports. The wire versions SHOULD be chosen at random, as the exposure of arbitrary version numbers prevents network

devices from incorrectly assuming that the version numbers are stable.

For each connection establishment attempt, a client SHOULD randomly choose one wire version, and the endpoints MUST use long header packets containing the chosen wire version throughout that connection establishment attempt.

2.3. Initial Packet

2.3.1. Mapping to Connections

A server associates an Initial packet to an existing connection using the Destination Connection ID, QUIC version, and the five tuple. If all of the values match to that of an existing connection, the packet is processed accordingly. Otherwise, a server MUST handle the packet as potentially creating a new connection.

2.3.2. 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, where Zx is the extracted DH shared secret of Encrypted SNI:

```
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.

2.3.3. Destination Connection ID

When establishing a connection, a client MUST initially set the Destination Connection ID to the hashed value of the first payload of the CRYPTO stream (i.e., the ClientHello message) truncated to first

sixteen (16) bytes. The hash function being used is the one selected by Encrypted SNI.

When processing the first payload carried by a CRYPTO stream, a server MUST, in addition to verifying the authentication tag, verify that the truncated hash value of the payload is identical to the Destination Connection ID or to the original Connection ID recovered from the the Retry Token. A server MUST NOT create or modify connection state if either or both the verification fails.

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.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+
|1|1| 3 |   0   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Version (32)                          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|DCIL(4)|SCIL(4)|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               Destination Connection ID (0/32..144)         ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                 Source Connection ID (0/32..144)            ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Error Code (16)     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

(: #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 one Initial packet in response to each Retry packet it receives. The Destination Connection ID of the Initial packet MUST be set to the value specified by the Retry packet, however the keys for encrypting and authenticating the packet MUST continue to be the original ones. A server sending a Retry packet is expected to include the original Connection ID in the Retry Token it emits, and to use the value contained in the token attached to the Initial packet for unprotecting the payload.

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 an Initial packet with the Destination Connection ID and the token set to the original value.

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. Split Mode

To support server-side deployments using "Split Mode" ([TLS-ESNI]; section 3), the following properties need to be exchanged between the fronting server and the hidden server, in addition to those generally required by a QUIC version 1 proxy and the Encrypted SNI extension:

o hmac_key

o ODCID

Both the fronting server and the hidden server need access to the hmac_key to authenticate the Initial packets. However, because the key is derived from the shared DH secret of ESNI, it is not necessarily available to the hidden server.

ODCID is necessary to decrypt an Initial packet sent in response to a Retry. However, the value is typically available only to the server that generates the Retry. The fronting server and the hidden server need to exchange the ODCID, or provide the secret for extracting the ODCID from a Retry token.

4. Security Considerations

The authenticated handshake is designed to enable successful connections even if clients and servers are attacked by a powerful "man on the side", which cannot delete packets but can inject packets and will always win the race against original packets.i We want to enable the following pattern: ```

Client Attacker Server

CInitial -> CInitial' -> CInitial -> <- SInitial <- SInitial' <- SInitial

CHandshake -> CHandshake -> ``` The goal is a successful handshake despite injection by the attacker of fake Client Initial packet (CInitial') or Server Initial packet (SInitial').

The main defense against forgeries is the HMAC authentication of the Initial packets using an ESNI derived key that is not accessible to

the attacker. This prevents all classes of attacks using forged Initial packets. There are however two methods that are still available to the attackers:

1) Forge an Initial packet that will claim the same context as the client request,

2) Send duplicates of the client request from a fake source address.

These two attacks and their mitigation are discussed in the next sections.

4.1. Resisting the duplicate context attack

The attacker mounts a duplicate context attack by observing the original Client Initial packet, and then creating its own Client Initial packet in which source and destination CID are the same as in the original packet. The ESNI secret will be different, because the packet is composed by the server. The goal of the attacker is to let the server create a context associated with the CID, so that when the original Client Initial later arrives it gets discarded.

This attack is mitigated by verifying that the Destination CID of the Client Initial matches the hash of the first CRYPTO stream payload.

If the server uses address verification, there may be a Retry scenario: ``` Client Attacker Server

CInitial -> <- Retry (with Token) CInitial2 (including Token) -> <- Sinitial

CHandshake -> ``` The Destination CID of the second Client Initial packet is selected by the server, or by a device acting on behalf of the server. This destination CID will not match the hash of the CRYPTO stream payload. However, in the retry scenario, the server is already rquired to know the Destination CID from the original Client Initial packet (ODCID), because it has to echo it in the transport parameters extension. The server can then verify that the hash of the CRYPTO stream payload matches the ODCID.

4.2. Resisting Address Substitution Attacks

The DCID of the original Initial packet is defined as the hash of the first payload of the CRYPTO stream. This prevents attackers from sending "fake" Initial packets that would be processed in the same server connection context as the authentic packet. However, it does not prevent address substitution attacks such as: ``` Client Attacker Server

CInitial(from A) -> CInitial(from A') -> CInitial(from A) -> ``` In this attack, the attacker races a copy of the Initial packet, substituting a faked value for the client's source address. The goal of the attack is to cause the server to associate the fake address with the connection context, causing the connection to fail.

The server cannot prevent this attack by just verifying the HMAC, because the address field is not covered by the checksum authentication. To actually mitigate the attack, the server needs to create different connection contexts for each pair of Initial DCID and source Address. The resulting exchange will be: ``` Client Attacker Server

CInitial(from A) -> CInitial(A') -> <- SInitial-X(to A') CInitial(A) -> <- SInitial-Y(to A) CHandshake-Y -> ```

The server behavior is required even if the server uses address verification procedures, because the attacker could mount a complex attack in which it obtains a Retry Token for its own address, then forwards it to the client: ``` Client Attacker Server

CInitial(from A) -> CInitial(from A') -> <- Retry(to A', T(A')) <- Retry(to A, T(A')) CInitial2(from A, T(A')) -> CInitial(from A', T(A')) ->

CInitial(from A) ->
<- Retry(T(A)) CInitial3(from A,
T(A)) -> ``` At the end of this exchange, the server will have received two valid client Initial packets that both path address verification and the ESNI based HMAC, and both have the same CRYPTO stream initial payload and the same ODCID. If it kept only one of them, the attacker would have succeeded in distrupting the connection attempt.

5. IANA Considerations

TBD

6. 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 TLS to Secure QUIC", draft-ietf-quic-tls-20 (work in progress), April 2019.

[QUIC-TRANSPORT]

Iyengar, J., Ed. and M. Thomson, Ed., "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-20 (work in progress), April 2019.

[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-03 (work in progress), March 2019.

Appendix A. Acknowledgements

TBD

Appendix B. Change Log

B.1. Since draft-kazuho-quic-authenticated-handshake-00

o Change DCID to Hash(ClientHello) (#8)

o Describe attacks (#12)

o Describe how Initial packets are mapped to connections (#10)

o Clarify the requirements to support split mode (#11)

o Version number greasing (#13)

Authors' Addresses

Kazuho Oku
Fastly

Email: kazuhooku@gmail.com

Christian Huitema
Private Octopus Inc.

Email: huitema@huitema.net