QUIC
Internet-Draft
Intended status: Informational
Expires: June 10, 2018

M. Thomson
Mozilla

December 07, 2017

More Apparent Randomization for QUIC
draft-thomson-quic-grease-00

Abstract

Options for creating more apparent randomization in the QUIC header are discussed.

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 https://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 June 10, 2018.

1. More Grease for QUIC

The QUIC invariants draft [INVARIANTS] commits QUIC [QUIC] to a small set of traits that are intended to remain stable across all version of QUIC. However, there are some protocol traits in QUIC version 1 [QUIC] that we do that remain readable.

This note explores a few options for protecting QUIC against casual inspection by entities other than the endpoints participating in the connection. These techniques are aimed especially at making any form of inspection considerably more difficult if the QUIC version of a packet is unknown.

The intent of applying this protection is to encourage the use of protocol fields that are intentionally designed to be readable to non-participating entities (see also [SIGNALS]). For those fields that can be recovered without access to negotiated cryptographic keys, the intent is to create an incentive to implement version-specific handling rather than to assume that certain properties don't change between versions.

2. Overall Design

Protocol fields that deploy with predictable values or a limited range of values can ossify. Ossification is the effect whereby the use of values in a way that is contradictory to established patterns triggers adverse reactions from the network. Usually, this is a result of middleboxes having developed assumptions about how protocols operate.

The idea that ossification actively prevents the deployment of modified protocols remains a little contentious in the community. On the other hand, we have plenty of evidence from TLS deployment to suggest that this happens. Appendix D.4 of [TLS13] describes an example of ossification and describes the measures that were necessary to counteract it.

If it is possible to provide a measure of protection against protocol ossification without inordinate expense, then it is the view of at least this author that doing so would have some potential value. Provided the costs are indeed low enough

This describes changes to some of the version-specific fields in QUIC version 1 [QUIC]. Any invariant [INVARIANTS] would not be affected by this change.

The simplest defense against ossification is to apply a reversible permutation to these values. A pseudo-random function (PRP) is the

obvious choice. If the key of that function is only known to endpoints, then the values will be readily accessible to endpoints.

3. Keying

There are two different contexts in which we might consider applying this sort of protection, and the keys we can use differ.

QUIC already has the tools necessary to derive keys. Handshake packets use a key that is derived from a combination of a version-specific key (or salt) and the connection ID. A similar approach could be used here.

For packets that have packet protection, there are many options available. The secret used for generating the packet number gap (packet_number_secret) is a candidate. It might however be better to derive a key from the packet protection secrets (client_pp_secret_<N> or server_pp_secret_<N>).

4. Pseudo-Random Permutations

There are many PRP functions that could apply to this case. Most come from cryptographic contexts and therefore assume inputs and state spaces that exceed the size of the fields we're interested in. This section presents three options of varying complexity. It's likely that there are many more.

It is critical to note that these functions are not intended to provide any real confidentiality - you need strong keys and ciphers for that. None of the functions include any integrity protection either - QUIC already provides integrity protection for its headers.

These techniques might be used to enable unlinkability in some circumstances.

4.1. Remainder

The first is a simple masking using a remainder remainder operation. This assumes that there are a limited number of valid values for a given field. Valid values are either constrained to or mapped to a contiguous range starting from 0. Then, select a modulus |m| that is greater than the largest value and encryption and decryption are simple:

```
m = max(x) + k
E(x) = x + random() * m
D(x) = x % m
```

The drawback here is that you don't get a uniform distribution. If we use this for the long header and an Initial packet is type 0, then it will never pick anything that isn't a multiple of |m|. Also, the number of values that |k| can assume is small. The benefit is that this is trivial to implement. Also, an implementation that chooses not to randomize still produces values that can be understood.

4.2. XOR

This PRP uses a simple exclusive OR:

D(x) = E(x) = x ^ k

This method is the easiest to implement. The drawback here is that |k| is stable and is therefore trivially recovered when multiple messages use the same key.

4.3. FFX Lite

FFX [1] is a mode of format-preserving encryption that encrypts values from a space of essentially arbitrary size. FFX would be ideal apart from one significant drawback: FFX is extremely computationally expensive for smaller values, as it uses more rounds for short values to ensure that it continues to preserve its security margins.

On the other hand, we're not looking for any actual security, so we wouldn't need to have the obscene number of rounds that FFX depends on for small values (their recommended parameters don't include values for 5 bit values,).

If you make a few choices (cryptographers rarely do this for you, and FFX has the usual cornucopia of tuning parameters), you can produce a set of parameters that makes FFX quite simple and performant. This is essentially two rounds of FFX with radix=2, P={}, addition=0, method=2, split(n)=floor(n/2), rnds(n)=2, and F(x)=AES(k, x).

```
E(x):
   split = num_bits(x) / 2
   a = x >> split
   b = s & (1 << split - 1)
   tmp = a ^ AES(k, b)
   result = (tmp << split) | (b ^ AES(k, tmp))

D(x):
   split = num_bits(x) / 2
   a = x >> split
   b = s & (1 << split - 1)
   tmp = b ^ AES(k, a)
   result = ((a ^ AES(k, tmp)) << split) | tmp
```

4.4. Predictability

The XOR and FFX-based methods exhibit properties similar to encryption with AES-ECB mode. That is, for a given key, the same plaintext will always encrypt to the same ciphertext.

That means that if input values do not vary over time, it will be possible to infer underlying values easily. The goal is to ensure that the protected values change for each new connection, for which a changing key is sufficient. As stated, the goal is not to provide confidentiality against a determined attacker, only to defend against a lazy observer.

5. Scrambling QUIC Packet Headers

There are two fields in the QUIC header that merit some degree of scrambling: the packet type and the packet number. The other fields of QUIC packets: invariant bits, connection ID, version, and the message payload are either:

- invariant and thus important to leave unmodified,
- explicitly designed for consumption by middleboxes and thus important to leave unmodified, or
- already protected by other means, such as an AEAD [AEAD].

It's possible that this technique could be applied to invariant fields, but that is likely to have less immediate utility. Worse, it would commit every future version of the protocol to employ the same technique. Limiting this to version-specific fields allows the technique to improve with successive protocol versions.

5.1. Scrambling Packet Types

We have two values here, a 7-bit value that is used for the long header, and a 5-bit value that is used for the short header. It is possible that the KEY_PHASE bit in the short header could also be covered.

Any of the described methods could work to obscure these fields. The remainder method would however degrade as new packet types are defined, though the current set of types is very small.

The cost of scrambling for packet types is that the entire space of values is then used, which could make multiplexing with realtime protocols more challenging. However, it is possible that those protocols could use their out-of-band negotiation to influence the scrambling. For instance, they might mandate the use of a connection ID from a set of values that produce values that are compatible with the multiplexing scheme in use (determining such a value would be easy).

5.2. Scrambling Packet Numbers

Scrambling packet numbers is relatively straightforward to apply. The remainder method doesn't work here, though the XOR and FFX-based techniques both work well.

The cost of scrambling the packet number is that it would make it more difficult to use packet numbers to support the use of other features, like the heuristics necessary to use the spin bit in the presence of loss and reordering.

If a variable-length integer is used to represent a packet number and FFX is chosen, the length would need to be separately scrambled. That suggests that retaining the packet number length in the type field is desirable if FFX is chosen.

For a packet number, the XOR-based technique would not provide any appreciable barrier to recovery of the underlying value.

5.2.1. Taking Packet Number Scrambling Further

If packet numbers are scrambled, it is possible to use that scrambling instead of both initial packet number randomization and the packet number gap.

For the initial packet number, scrambling would be sufficient to ensure that the packet number field could contain all possible values. That removes the need to reserve 1024 values to avoid

overflow of the 32-bit space before a peer receives the initial value. Packet numbers would always start at 0, but the wire encoding would be encoded.

For the packet number gap, if the key calculation takes connection ID as input, the need for a packet number gap is eliminated. Switching to a new connection ID would cause packet numbers to become unlinkable with previous ones. Deriving the per-connection-ID key with HKDF would ensure that even with a simple XOR, the two keys can't be correlated.

6. Security Considerations

This section exists so that I can submit a draft without being badgered about this. There are almost certainly security concerns here, but I don't care. This draft is a throwaway.

7. References

7.1. Informative References

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

[INVARIANTS] Thomson, M., "Version-Independent Properties of QUIC", draft-thomson-quic-invariants-00 (work in progress), November 2017.

[QUIC] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", draft-ietf-quic-transport-08 (work in progress), December 2017.

[SIGNALS] Hardie, T., "Path signals", draft-hardie-path-signals-02 (work in progress), November 2017.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-22 (work in progress), November 2017.

7.2. URIs

[1] https://csrc.nist.gov/CSRC/media/Projects/Block-Cipher-Techniques/documents/BCM/proposed-modes/ffx/ffx-spec.pdf

Author's Address

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com