Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 29, 2014

J. Snell

November 25, 2013

HTTP/2.0 Intra-Connection Negotiation
draft-snell-httpbis-keynego-02

Abstract

This memo describes a proposed modification to HTTP/2.0 that introduces the concepts of Intra-Connection Negotiation and Secure Framing.

Status of This Memo

This Internet-Draft is submitted to IETF 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 May 29, 2014.

Table of Contents

1. Introduction

HTTP/2.0 Intra-Connection Negotiation allows peers to dynamically negotiate agreements (e.g. for cryptographic keys) with an origin (as defined by [RFC6454]) from within an established HTTP/2.0 connection.

This mechanism would provide a number of important benefits, including:

1. The ability to negotiate multiple agreements for one or more origins within a single HTTP/2.0 connection;

2. The ability to revoke and renegotiate agreements on the fly without tearing down and reestablishing the HTTP/2.0 connection;

3. Support for multiple negotiation mechanisms, including pre-shared key, etc;

2. Agreement Negotiation

Intra-Connection Negotiation is facilitated through the use of a new "NEGOTIATE" HTTP pseudo-method. The HTTP header field mapping for the NEGOTIATE method works similarly to that of CONNECT methods, with a few notable exceptions:

- The ":method" header field is set to "NEGOTIATE".

- The ":scheme" and ":path" header fields MUST be omitted.

- The ":authority" header field contains the host and port of the origin for which an agreement is being negotiated.

- An ":id" header field MUST be given specifying the 31-bit numeric identifier of the agreement being negotiated.

- An ":algorithm" header field MUST be given specifying the IRI identifier of the negotiation / agreement algorithm being utilized.

A complete negotiation consumes a single stream within a connection and may consist of one or more distinct "messages" exchanged within that stream. The number of messages required for a negotiation depends on the specific algorithm being used. On HEADERS and DATA frames, the currently reserved 0x2 flag is used to signal the end of individual messages. The negotiation is considered complete when the stream is closed. A negotiated agreement cannot be used until the negotiation for is completed.

2.1. Example: Pre-Shared Key

To illustrate the basic flow of the negotiation protocol, consider the simple case where both peers share a common pre-shared secret. To simplify the example, we assume that there is need to prove possession of the shared secret.

The initiating peer would send:

```
HEADERS
  END_STREAM  (0x1)
  END_MESSAGE (0x2)
  END_HEADERS (0x4)
  :method = NEGOTIATE
  :authority = example.org
  :id = 1
  :algorithm = urn:example:algorithm:psk
  name = Our Shared Key Name
```

The flags END_STREAM and END_MESSAGE indicate to the receiving peer that no additional messages will be sent for this negotiation. Assuming the negotiation is accepted, a simplified response would be:

```
HEADERS
  END_STREAM  (0x1)
  END_MESSAGE (0x2)
  END_HEADERS (0x4)
  :status = 200
```

2.2. Example: Multi-step Negotiation

Some negotiation algorithms require multiple steps. This is accomplished by exchanging multiple messages within a single stream.

A "message" consists of a combination of a HEADERS frame followed by zero or more DATA frames. The last frame in the message MUST have the END_MESSAGE flag set.

Initializing a multi-step negotiation (note that the END_STREAM flag is not set)

```
HEADERS
  END_MESSAGE (0x2)
  END_HEADERS (0x4)
  :method = NEGOTIATE
  :authority = example.org
  :id = 1
  :algorithm = urn:example:algorithm:multistep
  init-param-1 = foo
```

The initializing reponse (again, note that the END_STREAM flag is not set)

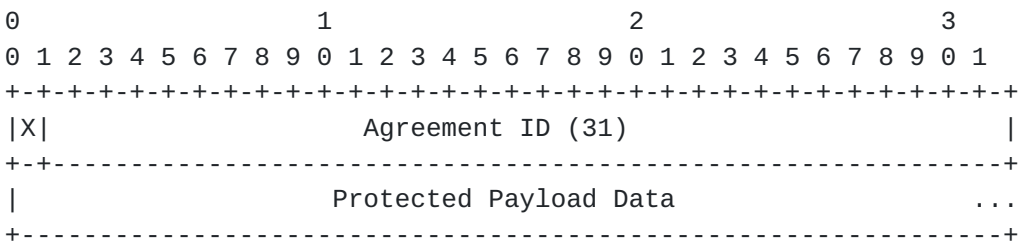
```
HEADERS
  END_MESSAGE (0x2)
  END_HEADERS (0x4)
  :status = 200
  init-param-A = bar
```

Once the initial HEADERS frames are sent, the peers are free to exchange as many messages on the stream as necessary to complete the negotiation process. When a peer is done with it's part of the negotiation, it will include the END_STREAM flag on the last frame it sends. If the negotiation process fails after the initial HEADERS frames are sent, an RST_STREAM frame is used to terminate the negotiation process.

3. Secure Framing (Option 1)

Obviously, negotiating an agreement is pointless if it cannot be subsequently used. To that end, I propose a modification to the existing DATA frame definition.

Specifically, I propose the introduction of a new AGREEMENT flag (0x4). When set, the flag indicates that the first four bytes of the DATA frame payload specify a numeric agreement identifier, and that the remaining DATA frame payload has been constructed in accordance with the referenced agreement. They specific structure of that data depends entirely on the properties of the agreement identified.



4. Secure Framing (Option 2)

A potential alternative (and likely better) option for use of a negotiated agreement is to move agreement identification out of the DATA frame and into a request header field. For instance:

```
HEADERS
  :method = POST
  :authority = example.org
  :agreement = 1
```

The presence of the ":agreement" header field in the initial HEADERS block indicates that all frames transmitted on the stream (in the same direction) are constructed in accordance to the specified agreement.

5. Renegotiation of Agreements

Depending on the nature of the agreement, it might be possible for the requesting peer to renegotiate an agreement with the origin. Significant care should be taken here, however, to prevent the possibility of downgrade attacks.

Renegotiation occurs by initiating a new NEGOTIATE request specifying an already established agreement identifier. This new interaction could establish new properties, expectations, etc for the agreement. The renegotiation is not complete until after both peers successfully close the stream, meaning any new negotiated properties do not become effective until after renegotiation is complete.

6. Explicit Termination of Agreements

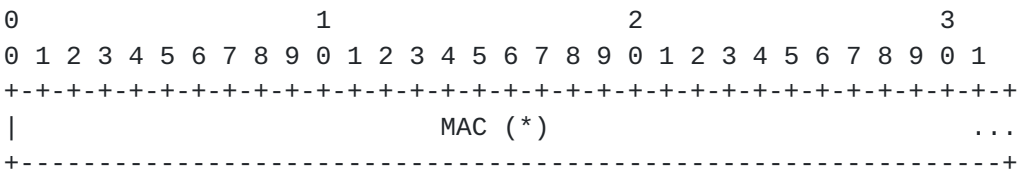
It is possible that a mechanism for explicitly revoking or terminating an agreement will be needed in some scenarios. Termination of an agreement is essentially a form of renegotiation and would happen following a similar approach. One possible method for terminating an agreement would be to send something like the following:

```
HEADERS
  :method = NEGOTIATE
  :id = 1
  :algorithm: urn:example:algorithm:revoke-agreement
```

A downside of this, however, termination would require action on the part of the requesting peer and could not be initiated by the origin unless we allow the origin to PUSH_PROMISE NEGOTIATE methods (which has it's own distinct problems since a client cannot send on a pushed stream).

7. The INTEGRITY frame type

The INTEGRITY frame (type=0xB) allows a sending peer to insert periodic message authentication codes (MACs) into a stream to provide integrity and authenticity of a streams content.



The INTEGRITY frame defines the following flags:

END_STREAM (0x1): Bit 1 being set indicates that this frame is the last that the endpoint will send for the identified stream. Setting this flag causes the stream to entire one of "half closed" states or "closed" state.

The payload of the INTEGRITY frame consists of a MAC calculated over the payloads of all other frames sent on a stream since either the stream was opened or a previous INTEGRITY frame was sent.

INTEGRITY frames MUST be associated with a stream that is, in turn, associated with a negotiated agreement. The algorithm used to gernerate the MAC is determined entirely through use of the NEGOTIATE pseudo-method.

8. Secure Tunneling with CONNECT

Obviously, the approach described thus far only secures the content of DATA frames. With HTTP, however, there is a significant amount of sensitive content carried within HEADERS frames. To provide a more complete solution, the mechanisms described herein can be combined with the CONNECT method to create a secure tunnel. Specifically:

- First, use the NEGOTIATE method to negotiate an agreement with an origin,

- Second, use the CONNECT method to establish a tunnel through that origin,

- Third, use SECURED DATA frames over the connected tunnel.

```
HEADERS
  END_STREAM
  :method = NEGOTIATE
  :authority = example.org
  :id = 1
...

HEADERS
  :method = CONNECT
  :authority = example.org
  :agreement = 1

DATA
  {Protected Data}

INTEGRITY
  {Mac}

DATA
  {Protected Data}

...
```

9. Security Considerations

TBD. TODO: Need to expand this...

10. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

Author's Address

James M Snell

Email: jasnell@gmail.com