HTTP
Internet-Draft
Intended status: Standards Track
Expires: April 26, 2019

M. Bishop
Akamai
N. Sullivan
Cloudflare
M. Thomson
Mozilla

October 23, 2018

Secondary Certificate Authentication in HTTP/2
draft-ietf-httpbis-http2-secondary-certs-03

Abstract

A use of TLS Exported Authenticators is described which enables HTTP/2 clients and servers to offer additional certificate-based credentials after the connection is established. The means by which these credentials are used with requests is defined.

Note to Readers

Discussion of this draft takes place on the HTTP working group mailing list (ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/ [1].

Working Group information can be found at http://httpwg.github.io/ [2]; source code and issues list for this draft can be found at https://github.com/httpwg/http-extensions/labels/secondary-certs [3].

Status of This Memo

Table of Contents

1. Introduction

HTTP clients need to know that the content they receive on a connection comes from the origin that they intended to retrieve in from. The traditional form of server authentication in HTTP has been in the form of a single X.509 certificate provided during the TLS ([RFC5246], [I-D.ietf-tls-tls13]) handshake.

Many existing HTTP [RFC7230] servers also have authentication requirements for the resources they serve. Of the bountiful authentication options available for authenticating HTTP requests, client certificates present a unique challenge for resource-specific authentication requirements because of the interaction with the underlying TLS layer.

TLS 1.2 [RFC5246] supports one server and one client certificate on a connection. These certificates may contain multiple identities, but only one certificate may be provided.

Many HTTP servers host content from several origins. HTTP/2 permits clients to reuse an existing HTTP connection to a server provided that the secondary origin is also in the certificate provided during the TLS handshake. In many cases, servers choose to maintain separate certificates for different origins but still desire the benefits of a shared HTTP connection.

1.1. Server Certificate Authentication

Section 9.1.1 of [RFC7540] describes how connections may be used to make requests from multiple origins as long as the server is authoritative for both. A server is considered authoritative for an origin if DNS resolves the origin to the IP address of the server and (for TLS) if the certificate presented by the server contains the origin in the Subject Alternative Names field.

[RFC7838] enables a step of abstraction from the DNS resolution. If both hosts have provided an Alternative Service at hostnames which resolve to the IP address of the server, they are considered authoritative just as if DNS resolved the origin itself to that address. However, the server's one TLS certificate is still required to contain the name of each origin in question.

[RFC8336] relaxes the requirement to perform the DNS lookup if already connected to a server with an appropriate certificate which claims support for a particular origin.

Servers which host many origins often would prefer to have separate certificates for some sets of origins. This may be for ease of certificate management (the ability to separately revoke or renew them), due to different sources of certificates (a CDN acting on behalf of multiple origins), or other factors which might drive this administrative decision. Clients connecting to such origins cannot currently reuse connections, even if both client and server would prefer to do so.

Because the TLS SNI extension is exchanged in the clear, clients might also prefer to retrieve certificates inside the encrypted context. When this information is sensitive, it might be advantageous to request a general-purpose certificate or anonymous ciphersuite at the TLS layer, while acquiring the "real" certificate in HTTP after the connection is established.

1.2. Client Certificate Authentication

For servers that wish to use client certificates to authenticate users, they might request client authentication during or immediately after the TLS handshake. However, if not all users or resources need certificate-based authentication, a request for a certificate has the unfortunate consequence of triggering the client to seek a certificate, possibly requiring user interaction, network traffic, or other time-consuming activities. During this time, the connection is stalled in many implementations. Such a request can result in a poor experience, particularly when sent to a client that does not expect the request.

The TLS 1.3 CertificateRequest can be used by servers to give clients hints about which certificate to offer. Servers that rely on certificate-based authentication might request different certificates for different resources. Such a server cannot use contextual information about the resource to construct an appropriate TLS CertificateRequest message during the initial handshake.

Consequently, client certificates are requested at connection establishment time only in cases where all clients are expected or required to have a single certificate that is used for all resources. Many other uses for client certificates are reactive, that is, certificates are requested in response to the client making a request.

1.2.1. HTTP/1.1 Using TLS 1.2 and Earlier

In HTTP/1.1, a server that relies on client authentication for a subset of users or resources does not request a certificate when the connection is established. Instead, it only requests a client certificate when a request is made to a resource that requires a certificate. TLS 1.2 [RFC5246] accommodates this by permitting the server to request a new TLS handshake, in which the server will request the client's certificate.

Figure 1 shows the server initiating a TLS-layer renegotiation in response to receiving an HTTP/1.1 request to a protected resource.

```
Client                                      Server
   -- (HTTP) GET /protected -------------------> *1
   <---------------------- (TLS) HelloRequest -- *2
   -- (TLS) ClientHello ---------------------->
   <------------------ (TLS) ServerHello, ... --
   <---------------- (TLS) CertificateRequest -- *3
   -- (TLS) ..., Certificate ------------------> *4
   -- (TLS) Finished ------------------------->
   <-------------------------- (TLS) Finished --
   <--------------------------- (HTTP) 200 OK -- *5
```

Figure 1: HTTP/1.1 reactive certificate authentication with TLS 1.2

In this example, the server receives a request for a protected resource (at *1 on Figure 1). Upon performing an authorization check, the server determines that the request requires authentication using a client certificate and that no such certificate has been provided.

The server initiates TLS renegotiation by sending a TLS HelloRequest (at *2). The client then initiates a TLS handshake. Note that some TLS messages are elided from the figure for the sake of brevity.

The critical messages for this example are the server requesting a certificate with a TLS CertificateRequest (*3); this request might use information about the request or resource. The client then provides a certificate and proof of possession of the private key in Certificate and CertificateVerify messages (*4).

When the handshake completes, the server performs any authorization checks a second time. With the client certificate available, it then authorizes the request and provides a response (*5).

1.2.2. HTTP/1.1 Using TLS 1.3

TLS 1.3 [I-D.ietf-tls-tls13] introduces a new client authentication mechanism that allows for clients to authenticate after the handshake has been completed. For the purposes of authenticating an HTTP request, this is functionally equivalent to renegotiation. Figure 2 shows the simpler exchange this enables.

```
Client                                      Server
   -- (HTTP) GET /protected ------------------->
   <---------------- (TLS) CertificateRequest --
   -- (TLS) Certificate, CertificateVerify,
               Finished ----------------------->
   <--------------------------- (HTTP) 200 OK --
```

Figure 2: HTTP/1.1 reactive certificate authentication with TLS 1.3

TLS 1.3 does not support renegotiation, instead supporting direct client authentication. In contrast to the TLS 1.2 example, in TLS 1.3, a server can simply request a certificate.

1.2.3. HTTP/2

An important part of the HTTP/1.1 exchange is that the client is able to easily identify the request that caused the TLS renegotiation. The client is able to assume that the next unanswered request on the connection is responsible. The HTTP stack in the client is then able to direct the certificate request to the application or component that initiated that request. This ensures that the application has the right contextual information for processing the request.

In HTTP/2, a client can have multiple outstanding requests. Without some sort of correlation information, a client is unable to identify which request caused the server to request a certificate.

Thus, the minimum necessary mechanism to support reactive certificate authentication in HTTP/2 is an identifier that can be use to correlate an HTTP request with a request for a certificate. Since streams are used for individual requests, correlation with a stream is sufficient.

[RFC7540] prohibits renegotiation after any application data has been sent. This completely blocks reactive certificate authentication in HTTP/2 using TLS 1.2. If this restriction were relaxed by an

extension or update to HTTP/2, such an identifier could be added to TLS 1.2 by means of an extension to TLS. Unfortunately, many TLS 1.2 implementations do not permit application data to continue during a renegotiation. This is problematic for a multiplexed protocol like HTTP/2.

1.3. HTTP-Layer Certificate Authentication

This draft defines HTTP/2 frames to carry the relevant certificate messages, enabling certificate-based authentication of both clients and servers independent of TLS version. This mechanism can be implemented at the HTTP layer without breaking the existing interface between HTTP and applications above it.

This could be done in a naive manner by replicating the TLS messages as HTTP/2 frames on each stream. However, this would create needless redundancy between streams and require frequent expensive signing operations. Instead, TLS Exported Authenticators [I-D.ietf-tls-exported-authenticator] are exchanged on stream zero and other frames incorporate them to particular requests by reference as needed.

TLS Exported Authenticators are structured messages that can be exported by either party of a TLS connection and validated by the other party. Given an established TLS connection, a request can be constructed which describes the desired certificate and an authenticator message can be constructed proving possession of a certificate and a corresponding private key. Both requests and authenticators can be generated by either the client or the server. Exported Authenticators use the message structures from Sections 4.3.2 and 4.4 of [I-D.ietf-tls-tls13], but different parameters.

Each Authenticator is computed using a Handshake Context and Finished MAC Key derived from the TLS session. The Handshake Context is identical for both parties of the TLS connection, while the Finished MAC Key is dependent on whether the Authenticator is created by the client or the server.

Successfully verified Authenticators result in certificate chains, with verified possession of the corresponding private key, which can be supplied into a collection of available certificates. Likewise, descriptions of desired certificates can be supplied into these collections.

Section 2 describes how the feature is employed, defining means to detect support in peers (Section 2.1), make certificates and requests available (Section 2.2), and indicate when streams are blocked waiting on an appropriate certificate (Section 2.3). Section 3

defines the required frame types, which parallel the TLS 1.3 message exchange. Finally, Section 4 defines new error types which can be used to notify peers when the exchange has not been successful.

1.4. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Discovering Additional Certificates at the HTTP/2 Layer

A certificate chain with proof of possession of the private key corresponding to the end-entity certificate is sent as a sequence of "CERTIFICATE" frames (see Section 3.4) on stream zero. Once the holder of a certificate has sent the chain and proof, this certificate chain is cached by the recipient and available for future use. Clients can proactively indicate the certificate they intend to use on each request using an unsolicited "USE_CERTIFICATE" frame, if desired. The previously-supplied certificates are available for reference without having to resend them.

Otherwise, the server uses a "CERTIFICATE_REQUEST" frame to describe a class of certificates on stream zero, then uses "CERTIFICATE_NEEDED" frames to associate these with individual requests. The client responds with a "USE_CERTIFICATE" frame indicating the certificate which should be used to satisfy the request.

Data sent by each peer is correlated by the ID given in each frame. This ID is unrelated to values used by the other peer, even if each uses the same ID in certain cases. "USE_CERTIFICATE" frames indicate whether they are sent proactively or are in response to a "CERTIFICATE_NEEDED" frame.

2.1. Indicating Support for HTTP-Layer Certificate Authentication

Clients and servers that will accept requests for HTTP-layer certificate authentication indicate this using the HTTP/2 "SETTINGS_HTTP_CERT_AUTH" (0xSETTING-TBD) setting.

The initial value for the "SETTINGS_HTTP_CERT_AUTH" setting is 0, indicating that the peer does not support HTTP-layer certificate authentication. If a peer does support HTTP-layer certificate authentication, the value is non-zero.

In order to ensure that the TLS connection is direct to the server, rather than via a TLS-terminating proxy, each side will separately compute and confirm the value of this setting. The setting is derived from a TLS exporter (see Section 7.5 of [I-D.ietf-tls-tls13] and [RFC5705] for more details on exporters). Clients MUST NOT use an early exporter during their 0-RTT flight, but MUST send an updated SETTINGS frame using a regular exporter after the TLS handshake completes.

The exporter is constructed with the following input:

- o Label:
  - * "EXPORTER HTTP CERTIFICATE client" for clients
  - * "EXPORTER HTTP CERTIFICATE server" for servers
- o Context: Empty
- o Length: Four bytes

The resulting exporter is converted to a setting value as:

```
(Exporter & 0x3fffffff) | 0x80000000
```

That is, the most significant bit will always be set, regardless of the value of the exporter. Each endpoint will compute the expected value from their peer. If the setting is not received, or if the value received is not the expected value, the frames defined in this document SHOULD NOT be sent.

2.2. Making Certificates or Requests Available

When both peers have advertised support for HTTP-layer certificates as in Section 2.1, either party can supply additional certificates into the connection at any time. This means that clients or servers which predict a certificate will be required could supply the certificate before being asked. These certificates are available for reference by future "USE_CERTIFICATE" frames.

Certificates supplied by servers can be considered by clients without further action by the server. A server SHOULD NOT send certificates which do not cover origins which it is prepared to service on the current connection, but MAY use the ORIGIN frame [RFC8336] to indicate that not all covered origins will be served.

```
   Client                                      Server
      <------------------ (stream 0) CERTIFICATE --
      ...
      -- (stream N) GET /from-new-origin --------->
      <----------------------- (stream N) 200 OK --
```

Figure 3: Proactive server authentication

```
   Client                                      Server
      -- (stream 0) CERTIFICATE ------------------>
      -- (stream 0) USE_CERTIFICATE (S=1) -------->
      -- (stream 0) USE_CERTIFICATE (S=3) -------->
      -- (streams 1,3) GET /protected ------------>
      <-------------------- (streams 1,3) 200 OK --
```

Figure 4: Proactive client authentication

Likewise, either party can supply a "CERTIFICATE_REQUEST" that outlines parameters of a certificate they might request in the future. Upon receipt of a "CERTIFICATE_REQUEST", endpoints SHOULD provide a corresponding certificate in anticipation of a request shortly being blocked. Clients MAY wait for a "CERTIFICATE_NEEDED" frame to assist in associating the certificate request with a particular HTTP transaction.

2.3. Requiring Certificate Authentication

2.3.1. Requiring Additional Server Certificates

As defined in [RFC7540], when a client finds that a https:// origin (or Alternative Service [RFC7838]) to which it needs to make a request has the same IP address as a server to which it is already connected, it MAY check whether the TLS certificate provided contains the new origin as well, and if so, reuse the connection.

If the TLS certificate does not contain the new origin, but the server has claimed support for that origin (with an ORIGIN frame, see [RFC8336]) and advertised support for HTTP-layer certificates (see Section 2.1), the client MAY send a "CERTIFICATE_REQUEST" frame describing the desired origin. The client then sends a "CERTIFICATE_NEEDED" frame for stream zero referencing the request, indicating that the connection cannot be used for that origin until the certificate is provided.

If the server does not have the desired certificate, it MUST send an Empty Authenticator, as described in Section 5 of

[I-D.ietf-tls-exported-authenticator], in a "CERTIFICATE" frame in response to the request, followed by a "USE_CERTIFICATE" frame for stream zero which references the Empty Authenticator. In this case, or if the server has not advertised support for HTTP-layer certificates, the client MUST NOT send any requests for resources in that origin on the current connection.

```
Client                                            Server
   <------------------------ (stream 0) ORIGIN --
   -- (stream 0) CERTIFICATE_REQUEST ---------->
   -- (stream 0) CERTIFICATE_NEEDED (S=0) ----->
   <------------------- (stream 0) CERTIFICATE --
   <-------- (stream 0) USE_CERTIFICATE (S=0) --
   -- (stream N) GET /from-new-origin --------->
   <------------------------ (stream N) 200 OK --
```

Figure 5: Client-requested certificate

If a client receives a "PUSH_PROMISE" referencing an origin for which it has not yet received the server's certificate, this is a fatal connection error (see section 8.2 of [RFC7540]). To avoid this, servers MUST supply the associated certificates before pushing resources from a different origin.

2.3.2. Requiring Additional Client Certificates

Likewise, the server sends a "CERTIFICATE_NEEDED" frame for each stream where certificate authentication is required. The client answers with a "USE_CERTIFICATE" frame indicating the certificate to use on that stream. If the request parameters or the responding certificate are not already available, they will need to be sent as described in Section 2.2 as part of this exchange.

```
Client                                            Server
   <---------- (stream 0) CERTIFICATE_REQUEST --
   ...
   -- (stream N) GET /protected --------------->
   <----- (stream 0) CERTIFICATE_NEEDED (S=N) --
   -- (stream 0) CERTIFICATE ------------------>
   -- (stream 0) USE_CERTIFICATE (S=N) -------->
   <------------------------ (stream N) 200 OK --
```

Figure 6: Reactive certificate authentication

If the client does not have the desired certificate, it instead sends an Empty Authenticator, as described in Section 5 of [I-D.ietf-tls-exported-authenticator], in a "CERTIFICATE" frame in response to the request, followed by a "USE_CERTIFICATE" frame which

references the Empty Authenticator. In this case, or if the client has not advertised support for HTTP-layer certificates, the server processes the request based solely on the certificate provided during the TLS handshake, if any. This might result in an error response via HTTP, such as a status code 403 (Not Authorized).

3. Certificates Frames for HTTP/2

The "CERTIFICATE_REQUEST" and "CERTIFICATE_NEEDED" frames are correlated by their "Request-ID" field. Subsequent "CERTIFICATE_NEEDED" frames with the same "Request-ID" value MAY be sent for other streams where the sender is expecting a certificate with the same parameters.

The "CERTIFICATE", and "USE_CERTIFICATE" frames are correlated by their "Cert-ID" field. Subsequent "USE_CERTIFICATE" frames with the same "Cert-ID" MAY be sent in response to other "CERTIFICATE_NEEDED" frames and refer to the same certificate.

"CERTIFICATE_NEEDED" and "USE_CERTIFICATE" frames are correlated by the Stream ID they reference. Unsolicited "USE_CERTIFICATE" frames are not responses to "CERTIFICATE_NEEDED" frames; otherwise, each "USE_CERTIFICATE" frame for a stream is considered to respond to a "CERTIFICATE_NEEDED" frame for the same stream in sequence.

```
+---------+           +---------+
| REQUEST |           |   CERT  |
+---------+           +---------+
     |                     |
     | Request-ID          | Cert-ID
     |                     |
     v                     v
+---------+ Stream ID +---------+
| NEEDED  |---------->|   USE   |
+---------+           +---------+
```

Figure 7: Frame correlation

"Request-ID" and "Cert-ID" are independent and sender-local. The use of the same value by the other peer or in the other context does not imply any correlation between these frames. These values MUST be unique per sender for each space over the lifetime of the connection.

3.1. The CERTIFICATE_NEEDED Frame

The "CERTIFICATE_NEEDED" frame (0xFRAME-TBD1) is sent on stream zero to indicate that the HTTP request on the indicated stream is blocked pending certificate authentication. The frame includes stream ID and

a request identifier which can be used to correlate the stream with a previous "CERTIFICATE_REQUEST" frame sent on stream zero. The "CERTIFICATE_REQUEST" describes the certificate the sender requires to make progress on the stream in question.

```
  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
 +-----------------------------------------------------------------+
 |R|                        Stream ID (31)                         |
 +-------------------------------+---------------------------------+
 |        Request-ID (16)        |
 +-------------------------------+
```

Figure 8: CERTIFICATE_NEEDED frame payload

The "CERTIFICATE_NEEDED" frame contains 6 octets. The first four octets indicate the Stream ID of the affected stream. The following two octets are the authentication request identifier, "Request-ID". A peer that receives a "CERTIFICATE_NEEDED" of any other length MUST treat this as a stream error of type "PROTOCOL_ERROR". Frames with identical request identifiers refer to the same "CERTIFICATE_REQUEST".

A server MAY send multiple "CERTIFICATE_NEEDED" frames for the same stream. If a server requires that a client provide multiple certificates before authorizing a single request, each required certificate MUST be indicated with a separate "CERTIFICATE_NEEDED" frame, each of which MUST have a different request identifier (referencing different "CERTIFICATE_REQUEST" frames describing each required certificate). To reduce the risk of client confusion, servers SHOULD NOT have multiple outstanding "CERTIFICATE_NEEDED" frames for the same stream at any given time.

Clients MUST only send multiple "CERTIFICATE_NEEDED" frames for stream zero. Multiple "CERTIFICATE_NEEDED" frames on any other stream MUST be considered a stream error of type "PROTOCOL_ERROR".

The "CERTIFICATE_NEEDED" frame MUST NOT be sent to a peer which has not advertised support for HTTP-layer certificate authentication.

The "CERTIFICATE_NEEDED" frame MUST NOT reference a stream in the "half-closed (local)" or "closed" states [RFC7540]. A client that receives a "CERTIFICATE_NEEDED" frame for a stream which is not in a valid state SHOULD treat this as a stream error of type "PROTOCOL_ERROR".

3.2. The USE_CERTIFICATE Frame

The "USE_CERTIFICATE" frame (0xFRAME-TBD4) is sent on stream zero to indicate which certificate is being used on a particular request stream.

The "USE_CERTIFICATE" frame defines a single flag:

UNSOLICITED (0x01): Indicates that no "CERTIFICATE_NEEDED" frame has yet been received for this stream.

The payload of the "USE_CERTIFICATE" frame is as follows:

```
  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
 +-----------------------------------------------------------------+
 |R|                        Stream ID (31)                         |
 +-------------------------------+-------------------------------+
 |         [Cert-ID (16)]        |
 +-------------------------------+
```

Figure 9: USE_CERTIFICATE frame payload

The first four octets indicate the Stream ID of the affected stream. The following two octets, if present, contain the two-octet "Cert-ID" of the certificate the sender wishes to use. This MUST be the ID of a certificate for which proof of possession has been presented in a "CERTIFICATE" frame. Recipients of a "USE_CERTIFICATE" frame of any other length MUST treat this as a stream error of type "PROTOCOL_ERROR". Frames with identical certificate identifiers refer to the same certificate chain.

A "USE_CERTIFICATE" frame which omits the Cert-ID refers to the certificate provided at the TLS layer, if any. If no certificate was provided at the TLS layer, the stream should be processed with no authentication, likely returning an authentication-related error at the HTTP level (e.g. 403) for servers or routing the request to a new connection for clients.

The "UNSOLICITED" flag MAY be set by clients on the first "USE_CERTIFICATE" frame referring to a given stream. This permits a client to proactively indicate which certificate should be used when processing a new request. When such an unsolicited indication refers to a request that has not yet been received, servers SHOULD cache the indication briefly in anticipation of the request.

Receipt of more than one unsolicited "USE_CERTIFICATE" frames or an unsolicited "USE_CERTIFICATE" frame which is not the first in

reference to a given stream MUST be treated as a stream error of type "CERTIFICATE_OVERUSED".

Each "USE_CERTIFICATE" frame which is not marked as unsolicited is considered to respond in order to the "CERTIFICATE_NEEDED" frames for the same stream. If a "USE_CERTIFICATE" frame is received for which a "CERTIFICATE_NEEDED" frame has not been sent, this MUST be treated as a stream error of type "CERTIFICATE_OVERUSED".

Receipt of a "USE_CERTIFICATE" frame with an unknown "Cert-ID" MUST result in a stream error of type "PROTOCOL_ERROR".

The referenced certificate chain needs to conform to the requirements expressed in the "CERTIFICATE_REQUEST" to the best of the sender's ability, or the recipient is likely to reject it as unsuitable despite properly validating the authenticator. If the recipient considers the certificate unsuitable, it MAY at its discretion either return an error at the HTTP semantic layer, or respond with a stream error [RFC7540] on any stream where the certificate is used. Section 4 defines certificate-related error codes which might be applicable.

3.3. The CERTIFICATE_REQUEST Frame

The "CERTIFICATE_REQUEST" frame (id=0xFRAME-TBD2) provides a exported authenticator request message from the TLS layer that specifies a desired certificate. This describes the certificate the sender wishes to have presented.

The "CERTIFICATE_REQUEST" frame SHOULD NOT be sent to a peer which has not advertised support for HTTP-layer certificate authentication.

The "CERTIFICATE_REQUEST" frame MUST be sent on stream zero. A "CERTIFICATE_REQUEST" frame received on any other stream MUST be rejected with a stream error of type "PROTOCOL_ERROR".

```
  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
 +-------------------------------+-------------------------------+
 |        Request-ID (16)        |          Request (?)        ...
 +---------------------------------------------------------------+
```

Figure 10: CERTIFICATE_REQUEST frame payload

The frame contains the following fields:

Request-ID: "Request-ID" is a 16-bit opaque identifier used to correlate subsequent certificate-related frames with this request. The identifier MUST be unique in the session for the sender.

Request: An exported authenticator request, generated using the "request" API described in [I-D.ietf-tls-exported-authenticator]. See Section 3.4.1 for more details on the input to this API.

3.3.1. Exported Authenticator Request Characteristics

The Exported Authenticator "request" API defined in [I-D.ietf-tls-exported-authenticator] takes as input a set of desired certificate characteristics and a "certificate_request_context", which needs to be unpredictable. When generating exported authenticators for use with this extension, the "certificate_request_context" MUST contain both the two-octet Request-ID as well as at least 96 bits of additional entropy.

The TLS library on the authenticating peer will provide mechanisms to select an appropriate certificate to respond to the transported request. TLS libraries on servers MUST be able to recognize the "server_name" extension ([RFC6066]) at a minimum. Clients MUST always specify the desired origin using this extension, though other extensions MAY also be included.

3.4. The CERTIFICATE Frame

The "CERTIFICATE" frame (id=0xFRAME-TBD3) provides a exported authenticator message from the TLS layer that provides a chain of certificates, associated extensions and proves possession of the private key corresponding to the end-entity certificate.

The "CERTIFICATE" frame defines two flags:

TO_BE_CONTINUED (0x01): Indicates that the exported authenticator spans more than one frame.

```
  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
 +-------------------------------+-------------------------------+
 |          Cert-ID (16)         |   Authenticator Fragment (*)...
 +---------------------------------------------------------------+
```

Figure 11: CERTIFICATE frame payload

The frame contains the following fields:

Cert-ID: "Cert-ID" is a 16-bit opaque identifier used to correlate other certificate- related frames with this exported authenticator fragment.

Authenticator Fragment: A portion of the opaque data returned from the TLS connection exported authenticator "authenticate" API. See Section 3.4.1 for more details on the input to this API.

An exported authenticator is transported in zero or more "CERTIFICATE" frames with the "TO_BE_CONTINUED" flag set, followed by one "CERTIFICATE" frame with the "TO_BE_CONTINUED" flag unset. Each of these frames contains the same "Cert-ID" field, permitting them to be associated with each other. Receipt of any "CERTIFICATE" frame with the same "Cert-ID" following the receipt of a "CERTIFICATE" frame with "TO_BE_CONTINUED" unset MUST be treated as a connection error of type "PROTOCOL_ERROR".

Upon receiving a complete series of "CERTIFICATE" frames, the receiver may validate the Exported Authenticator value by using the exported authenticator API. This returns either an error indicating that the message was invalid, or the certificate chain and extensions used to create the message.

The "CERTIFICATE" frame MUST be sent on stream zero. A "CERTIFICATE" frame received on any other stream MUST be rejected with a stream error of type "PROTOCOL_ERROR".

3.4.1. Exported Authenticator Characteristics

The Exported Authenticator API defined in [I-D.ietf-tls-exported-authenticator] takes as input a request, a set of certificates, and supporting information about the certificate (OCSP, SCT, etc.). The result is an opaque token which is used when generating the "CERTIFICATE" frame.

Upon receipt of a "CERTIFICATE" frame, an endpoint MUST perform the following steps to validate the token it contains:

- Using the "get context" API, retrieve the "certificate_request_context" used to generate the authenticator, if any.

- Verify that the "certificate_request_context" is either empty (clients only) or contains the Request-ID of a previously-sent "CERTIFICATE_REQUEST" frame.

- Use the "validate" API to confirm the validity of the authenticator with regard to the generated request (if any).

Once the authenticator is accepted, the endpoint can perform any other checks for the acceptability of the certificate itself.

4. Indicating Failures During HTTP-Layer Certificate Authentication

Because this draft permits certificates to be exchanged at the HTTP framing layer instead of the TLS layer, several certificate-related errors which are defined at the TLS layer might now occur at the HTTP framing layer. In this section, those errors are restated and added to the HTTP/2 error code registry.

BAD_CERTIFICATE (0xERROR-TBD1): A certificate was corrupt, contained signatures that did not verify correctly, etc.

UNSUPPORTED_CERTIFICATE (0xERROR-TBD2): A certificate was of an unsupported type or did not contain required extensions

CERTIFICATE_REVOKED (0xERROR-TBD3): A certificate was revoked by its signer

CERTIFICATE_EXPIRED (0xERROR-TBD4): A certificate has expired or is not currently valid

CERTIFICATE_GENERAL (0xERROR-TBD5): Any other certificate-related error

CERTIFICATE_OVERUSED (0xERROR-TBD6): More certificates were used on a request than were requested

As described in [RFC7540], implementations MAY choose to treat a stream error as a connection error at any time. Of particular note, a stream error cannot occur on stream 0, which means that implementations cannot send non-session errors in response to "CERTIFICATE_REQUEST", and "CERTIFICATE" frames. Implementations which do not wish to terminate the connection MAY either send relevant errors on any stream which references the failing certificate in question or process the requests as unauthenticated and provide error information at the HTTP semantic layer.

5. Security Considerations

This mechanism defines an alternate way to obtain server and client certificates other than in the initial TLS handshake. While the signature of exported authenticator values is expected to be equally secure, it is important to recognize that a vulnerability in this code path is at least equal to a vulnerability in the TLS handshake.

5.1. Impersonation

This mechanism could increase the impact of a key compromise. Rather than needing to subvert DNS or IP routing in order to use a compromised certificate, a malicious server now only needs a client to connect to _some_ HTTPS site under its control in order to present the compromised certificate. As recommended in [RFC8336], clients opting not to consult DNS ought to employ some alternative means to increase confidence that the certificate is legitimate.

As noted in the Security Considerations of [I-D.ietf-tls-exported-authenticator], it difficult to formally prove that an endpoint is jointly authoritative over multiple certificates, rather than individually authoritative on each certificate. As a result, clients MUST NOT assume that because one origin was previously collocated with another, those origins will be reachable via the same endpoints in the future. Clients MUST NOT consider previous secondary certificates to be validated after TLS session resumption. However, clients MAY proactively query for previously-presented secondary certificates.

5.2. Fingerprinting

This draft defines a mechanism which could be used to probe servers for origins they support, but opens no new attack versus making repeat TLS connections with different SNI values. Servers SHOULD impose similar denial-of-service mitigations (e.g. request rate limits) to "CERTIFICATE_REQUEST" frames as to new TLS connections.

While the extensions in the "CERTIFICATE_REQUEST" frame permit the sender to enumerate the acceptable Certificate Authorities for the requested certificate, it might not be prudent (either for security or data consumption) to include the full list of trusted Certificate Authorities in every request. Senders, particularly clients, SHOULD send only the extensions that narrowly specify which certificates would be acceptable.

5.3. Denial of Service

Failure to provide a certificate on a stream after receiving "CERTIFICATE_NEEDED" blocks processing, and SHOULD be subject to standard timeouts used to guard against unresponsive peers.

Validating a multitude of signatures can be computationally expensive, while generating an invalid signature is computationally cheap. Implementations will require checks for attacks from this direction. Invalid exported authenticators SHOULD be treated as a session error, to avoid further attacks from the peer, though an

implementation MAY instead disable HTTP-layer certificates for the current connection instead.

5.4. Confusion About State

Implementations need to be aware of the potential for confusion about the state of a connection. The presence or absence of a validated certificate can change during the processing of a request, potentially multiple times, as "USE_CERTIFICATE" frames are received. A server that uses certificate authentication needs to be prepared to reevaluate the authorization state of a request as the set of certificates changes.

Client implementations need to carefully consider the impact of setting the "AUTOMATIC_USE" flag. This flag is a performance optimization, permitting the client to avoid a round-trip on each request where the server checks for certificate authentication. However, once this flag has been sent, the client has zero knowledge about whether the server will use the referenced cert for any future request, or even for an existing request which has not yet completed. Clients MUST NOT set this flag on any certificate which is not appropriate for currently-in-flight requests, and MUST NOT make any future requests on the same connection which they are not willing to have associated with the provided certificate.

6. IANA Considerations

This draft adds entries in three registries.

The HTTP/2 "SETTINGS_HTTP_CERT_AUTH" setting is registered in Section 6.1. Four frame types are registered in Section 6.2. Six error codes are registered in Section 6.3.

6.1. HTTP/2 SETTINGS_HTTP_CERT_AUTH Setting

The SETTINGS_HTTP_CERT_AUTH setting is registered in the "HTTP/2 Settings" registry established in [RFC7540].

Name: SETTINGS_HTTP_CERT_AUTH

Code: 0xSETTING-TBD

Initial Value: 0

Specification: This document.

6.2. New HTTP/2 Frames

Four new frame types are registered in the "HTTP/2 Frame Types" registry established in [RFC7540]. The entries in the following table are registered by this document.

Frame Type	Code	Specification
CERTIFICATE_NEEDED	0xFRAME-TBD1	Section 3.1
CERTIFICATE_REQUEST	0xFRAME-TBD2	Section 3.3
CERTIFICATE	0xFRAME-TBD3	Section 3.4
USE_CERTIFICATE	0xFRAME-TBD4	Section 3.2

6.3. New HTTP/2 Error Codes

Six new error codes are registered in the "HTTP/2 Error Code" registry established in [RFC7540]. The entries in the following table are registered by this document.

Name	Code	Specification
BAD_CERTIFICATE	0xERROR-TBD1	Section 4
UNSUPPORTED_CERTIFICATE	0xERROR-TBD2	Section 4
CERTIFICATE_REVOKED	0xERROR-TBD3	Section 4
CERTIFICATE_EXPIRED	0xERROR-TBD4	Section 4
CERTIFICATE_GENERAL	0xERROR-TBD5	Section 4
CERTIFICATE_OVERUSED	0xERROR-TBD6	Section 4

7. References

7.1. Normative References

[I-D.ietf-tls-exported-authenticator]
Sullivan, N., "Exported Authenticators in TLS", draft-ietf-tls-exported-authenticator-07 (work in progress), June 2018.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-28 (work in progress), March 2018.

[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>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[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, <https://www.rfc-editor.org/info/rfc7230>.

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

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

7.2. Informative References

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/info/rfc5705>.

[RFC7838] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/info/rfc7838>.

[RFC8336] Nottingham, M. and E. Nygren, "The ORIGIN HTTP/2 Frame", RFC 8336, DOI 10.17487/RFC8336, March 2018, <https://www.rfc-editor.org/info/rfc8336>.

7.3. URIs

[1] https://lists.w3.org/Archives/Public/ietf-http-wg/

[2] http://httpwg.github.io/

[3] https://github.com/httpwg/http-extensions/labels/secondary-certs

Appendix A. Change Log

RFC Editor's Note: Please remove this section prior to publication of a final version of this document.

A.1. Since draft-ietf-httpbis-http2-secondary-certs-01:

o Clients can send "CERTIFICATE_NEEDED" for stream 0 rather than speculatively reserving a stream for an origin.

o Use SETTINGS to disable when a TLS-terminating proxy is present (#617,#651)

A.2. Since draft-ietf-httpbis-http2-secondary-certs-00:

o All frames sent on stream zero; replaced "AUTOMATIC_USE" on "CERTIFICATE" with "UNSOLICITED" on "USE_CERTIFICATE". (#482,#566)

o Use Exported Requests from the TLS Exported Authenticators draft; eliminate facilities for expressing certificate requirements in "CERTIFICATE_REQUEST" frame. (#481)

A.3. Since draft-bishop-httpbis-http2-additional-certs-05:

o Adopted as draft-ietf-httpbis-http2-secondary-certs

Acknowledgements

Eric Rescorla pointed out several failings in an earlier revision. Andrei Popov contributed to the TLS considerations.

A substantial portion of Mike's work on this draft was supported by Microsoft during his employment there.

Authors' Addresses

Mike Bishop
Akamai

Email: mbishop@evequefou.be

Nick Sullivan
Cloudflare

Email: nick@cloudflare.com

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com