HTTP
Internet-Draft
Intended status: Standards Track
Expires: September 15, 2016

M. Thomson
Mozilla
M. Bishop
Microsoft

March 14, 2016

Reactive Certificate-Based Client Authentication in HTTP/2
draft-thomson-http2-client-certs-02

Abstract

Some HTTP servers provide a subset of resources that require additional authentication to interact with. HTTP/1.1 servers rely on TLS renegotiation that is triggered by a request to a protected resource. HTTP/2 made this pattern impossible by forbidding the use of TLS renegotiation. While TLS 1.3 provides an alternate mechanism to obtain client certificates, this mechanism does not map well to usage in TLS 1.2.

This document describes a how client authentication might be requested by a server as a result of receiving a request to a protected resource.

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 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 September 15, 2016.

Table of Contents

1. Introduction 3
1.1. Reactive Certificate Authentication in HTTP/1.1 4
1.1.1. Using TLS 1.2 and previous 4
1.1.2. Using TLS 1.3 5
1.2. Reactive Client Authentication in HTTP/2 5
1.3. Terminology 7
2. Presenting Client Certificates at the HTTP/2 Framing Layer . 7
2.1. The CERTIFICATE_REQUIRED frame 7
2.2. The USE_CERTIFICATE Frame 8
2.3. The CERTIFICATE_REQUEST Frame 9
2.4. The CERTIFICATE frame 10
2.5. The CERTIFICATE_PROOF Frame 12
3. Indicating failures during HTTP-Layer Certificate Authentication 13
4. Indicating Support for HTTP-Layer Certificate Authentication 14
5. Security Considerations 14
6. IANA Considerations 15
6.1. HTTP/2 SETTINGS_HTTP_CERT_AUTH Setting 16
6.2. New HTTP/2 Frames 16
6.2.1. CERTIFICATE_REQUIRED 16
6.2.2. CERTIFICATE_REQUEST 16
6.2.3. CERTIFICATE 16
6.2.4. CERTIFICATE_PROOF 16
6.2.5. USE_CERTIFICATE 17
6.3. New HTTP/2 Error Codes 17
6.3.1. BAD_CERTIFICATE 17
6.3.2. UNSUPPORTED_CERTIFICATE 17
6.3.3. CERTIFICATE_REVOKED 17
6.3.4. CERTIFICATE_EXPIRED 17
6.3.5. BAD_SIGNATURE 18
6.3.6. CERTIFICATE_GENERAL 18
7. Acknowledgements 18
8. Normative References 18
Authors' Addresses 19

1. Introduction

Many existing HTTP [RFC7230] servers have different authentication requirements for the different 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 RFC5246 [I-D.ietf-tls-tls13] layer.

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. 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.

In Yokohama, there was extensive working group discussion regarding why certificate authentication could not easily be done at the HTTP semantic layer. However, in subsequent discussion, it became apparent that the HTTP _framing_ layer did not suffer from the same limitation.

In this document, a mechanism for doing certificate-based client authentication via HTTP/2 frames is defined. This mechanism can be implemented at the HTTP layer without requiring new TLS stack behavior and without breaking the existing interface between HTTP and applications which employ client certificates.

1.1. Reactive Certificate Authentication in HTTP/1.1

1.1.1. Using TLS 1.2 and previous

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] accomodates 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.

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).

1.1.2. 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.

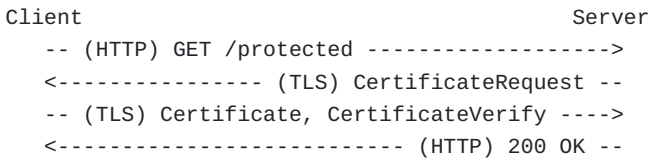

Figure 2: HTTP/1.1 Reactive Certificate Authentication with TLS 1.3

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. Reactive Client Authentication in 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.

Such an identifier could be added to TLS 1.2 by means of an extension, but 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. Instead, this draft proposes bringing the TLS 1.3 CertificateRequest, Certificate, and CertificateVerify messages into HTTP/2 frames, making client certificate authentication TLS-version-agnostic.

This could be done in a naive manner by replicating the messages as HTTP/2 frames on each stream. However, this would create needless

redundancy between streams and require frequent expensive signing operations. Instead, this draft lifts the bulky portions of each message into frames on stream zero and permits the on-stream frames to incorporate them by reference as needed.

On each stream where certificate authentication is required, the server sends a "CERTIFICATE_REQUIRED" frame, which the client answers with a "USE_CERTIFICATE" frame either indicating the certificate to use, or indicating that no certificate should be used. These frames are simple, referencing information previously sent on stream zero to reduce redundancy.

"CERTIFICATE_REQUIRED" frames reference a "CERTIFICATE_REQUEST" on stream zero, analogous to the CertificateRequest message. "USE_CERTIFICATE" frames reference a sequence of "CERTIFICATE" and "CERTIFICATE_PROOF" frames on stream zero, analogous to the the Certificate and CertificateVerify messages.

The exchange then looks like this:

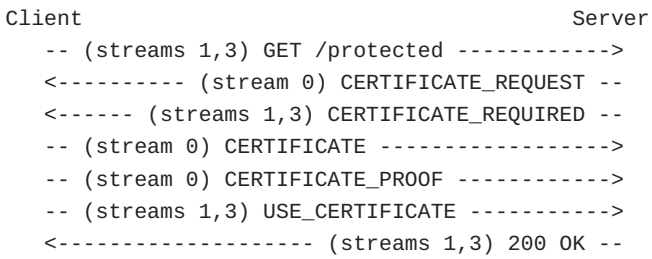

Figure 3: HTTP/2 Reactive Certificate Authentication

To avoid the extra round-trip per stream required for a challenge and response, the "AUTOMATIC_USE" flag enables a certificate to be automatically used by the server on subsequent requests without sending a "CERTIFICATE_REQUIRED" exchange.

Section 2 describes how certificates can be requested and presented at the HTTP/2 framing layer using several new frame types which parallel the TLS 1.3 message exchange. Section 3 defines new error types which can be used to notify peers when the exchange has not been successful. Finally, Section 4 describes how an HTTP/2 client can announce support for this feature so that a server might use these capabilities.

1.3. Terminology

RFC 2119 [RFC2119] defines the terms "MUST", "MUST NOT", "SHOULD" and "MAY".

2. Presenting Client Certificates at the HTTP/2 Framing Layer

An HTTP/2 request from a client that has signaled support for reactive certificate authentication (see Section 4) might cause a server to request client authentication. In HTTP/2 a server does this by sending at least one "CERTIFICATE_REQUEST" frame (see Section 2.3) on stream zero and sending a "CERTIFICATE_REQUIRED" frame (see Section 2.1) on the affected stream(s). The "CERTIFICATE_REQUEST" and "CERTIFICATE_REQUIRED" frames are correlated by their "Request-ID" field. Subsequent "CERTIFICATE_REQUIRED" frames with the same Request-ID MAY be sent on other streams where the server is expecting client authentication with the same parameters.

A server MAY send multiple concurrent "CERTIFICATE_REQUIRED" frames on the same stream. If a server requires that a client provide multiple certificates before authorizing a single request, it MUST send a "CERTIFICATE_REQUIRED" frame with a different request identifier and a corresponding "CERTIFICATE_REQUEST" frame describing each required certificate.

Clients respond to requests by sending one or more "CERTIFICATE" frames (see Section 2.4), followed by a "CERTIFICATE_PROOF" frame (see Section 2.5), on stream zero containing the "Request-ID" to which they are responding. The "USE_CERTIFICATE" (see Section 2.2) frame is sent on-stream to notify the server the stream is ready to be processed.

To reduce round-trips, the client MAY set the "AUTOMATIC_USE" flag on a "CERTIFICATE_PROOF" frame, indicating that the server SHOULD automatically apply the supplied certificate to any future streams matching that request, rather than sending a "CERTIFICATE_REQUIRED" frame.

2.1. The CERTIFICATE_REQUIRED frame

The "CERTIFICATE_REQUIRED" frame (0xFRAME-TBD2) is sent by servers to indicate that processing of an HTTP request is blocked pending certificate authentication. The frame includes a request identifier which can be used to correlate the stream with a previous "CERTIFICATE_REQUEST" frame received on stream zero. The "CERTIFICATE_REQUEST" describes the client certificate the server requires to process the request.

The "CERTIFICATE_REQUIRED" frame contains 1 octet, which is the authentication request identifier. A client that receives a "CERTIFICATE_REQUIRED" 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".

The "CERTIFICATE_REQUIRED" frame MUST NOT be sent by clients. A "CERTIFICATE_REQUIRED" frame received by a server SHOULD be rejected with a stream error of type PROTOCOL_ERROR.

The server MUST NOT send a "CERTIFICATE_REQUIRED" frame on stream zero, a server-initiated stream or a stream that does not have an outstanding request. In other words, a server can only send in the "open" or "half-closed (remote)" stream states.

A client that receives a "CERTIFICATE_REQUIRED" frame on a stream which is not in a valid state ("open" or "half-closed (local)" for clients) SHOULD treat this as a connection error of type "PROTOCOL_ERROR".

2.2. The USE_CERTIFICATE Frame

The "USE_CERTIFICATE" frame (0xFRAME-TBD5) is sent by clients in response to a "CERTIFICATE_REQUIRED" frame to indicate that the requested certificate has been provided (or will not be).

A "USE_CERTIFICATE" frame with no payload expresses the client's refusal to use the associated certificate (if any) with this stream. If the request was originally issued for a different stream, servers MAY create a new "CERTIFICATE_REQUEST" and permit the client to offer a different certificate. Alternatively, servers MAY process the request as unauthenticated, likely returning an authentication-related error at the HTTP level (e.g. 403).

Otherwise, the "USE_CERTIFICATE" frame contains the "Request-ID" of the now-completed certificate request. This MUST be an ID previously issued by the server, and for which a matching certificate has previously been presented along with a supporting certificate chain in one or more "CERTIFICATE" frames, and for which proof of possession has been presented in a "CERTIFICATE_PROOF" frame.

Use of the "USE_CERTIFICATE" frame by servers is not defined by this document. A "USE_CERTIFICATE" frame received by a client MUST be ignored.

The client MUST NOT send a "USE_CERTIFICATE" frame on stream zero, a server-initiated stream or a stream that does not have an outstanding request. In other words, a client can only send in the "open" or

"half-closed (local)" stream states. The client MUST NOT send a "USE_CERTIFICATE" frame except in response to a "CERTIFICATE_REQUIRED" frame from the server.

A server that receives a "USE_CERTIFICATE" frame on a stream which is not in a valid state ("open" or "half-closed (remote)" for servers), on which it has not sent a "CERTIFICATE_REQUIRED" frame, or referencing a certificate it has not previously received SHOULD treat this as a connection error of type "PROTOCOL_ERROR".

2.3. The CERTIFICATE_REQUEST Frame

TLS 1.3 defines the "CertificateRequest" message, which prompts the client to provide a certificate which conforms to certain properties specified by the server. This draft defines the "CERTIFICATE_REQUEST" frame (0xFRAME-TBD1), which contains the same contents as a TLS 1.3 "CertificateRequest" message, but can be sent over any TLS version.

The "CERTIFICATE_REQUEST" frame MUST NOT be sent by clients. A "CERTIFICATE_REQUEST" frame received by a server SHOULD be rejected with a stream error of type "PROTOCOL_ERROR".

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".

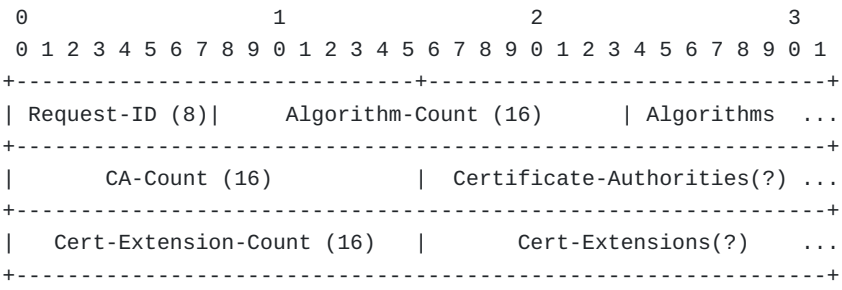

Figure 4: CERTIFICATE_REQUEST frame payload

The frame contains the following fields:

Request-ID: "Request-ID" is an 8-bit opaque identifier used to correlate subsequent certificate-related frames with this request. The identifier MUST be unique in the session.

Algorithm-Count and Algorithms: A list of the hash/signature algorithm pairs that the server is able to verify, listed in descending order of preference. Any certificates provided by the

client MUST be signed using a hash/signature algorithm pair found in "Algorithms". Each algorithm pair is encoded as a "SignatureAndHashAlgorithm" (see [I-D.ietf-tls-tls13] section 6.3.2.1), and the number of such structures is given by the 16-bit "Algorithm-Count" field, which MUST NOT be zero.

CA-Count and Certificate-Authorities: "Certificate-Authorities" is a series of distinguished names of acceptable certificate authorities, represented in DER-encoded [X690] format. These distinguished names may specify a desired distinguished name for a root CA or for a subordinate CA; thus, this message can be used to describe known roots as well as a desired authorization space. The number of such structures is given by the 16-bit "CA-Count" field, which MAY be zero. If the "CA-Count" field is zero, then the client MAY send any certificate that meets the rest of the selection criteria in the "CERTIFICATE_REQUEST", unless there is some external arrangement to the contrary.

Cert-Extension-Count and Cert-Extensions: A list of certificate extension OIDs [RFC5280] with their allowed values, represented in a series of "CertificateExtension" structures (see [I-D.ietf-tls-tls13] section 6.3.5). The list of OIDs MUST be used in certificate selection as described in [I-D.ietf-tls-tls13]. The number of Cert-Extension structures is given by the 16-bit "Cert-Extension-Count" field, which MAY be zero.

Some certificate extension OIDs allow multiple values (e.g. Extended Key Usage). If the sender has included a non-empty certificate_extensions list, the certificate MUST contain all of the specified extension OIDs that the recipient recognizes. For each extension OID recognized by the recipient, all of the specified values MUST be present in the certificate (but the certificate MAY have other values as well). However, the recipient MUST ignore and skip any unrecognized certificate extension OIDs.

PKIX RFCs define a variety of certificate extension OIDs and their corresponding value types. Depending on the type, matching certificate extension values are not necessarily bitwise-equal. It is expected that implementations will rely on their PKI libraries to perform certificate selection using these certificate extension OIDs.

2.4. The CERTIFICATE frame

A certificate chain is transferred as a series of "CERTIFICATE" frames (0xFRAME-TBD3) with the same Request-ID, each containing a single certificate in the chain. The end certificate of the chain can be used as authentication for previous or subsequent requests.

The "CERTIFICATE" frame defines no flags.

While unlikely, it is possible that an exceptionally large certificate might be too large to fit in a single HTTP/2 frame (see [RFC7540] section 4.2). Senders unable to transfer a requested certificate due to the recipient's "SETTINGS_MAX_FRAME_SIZE" value SHOULD terminate affected streams with "CERTIFICATE_TOO_LARGE".

Use of the "CERTIFICATE" frame by servers is not defined by this document. A "CERTIFICATE" frame received by a client MUST be ignored.

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".

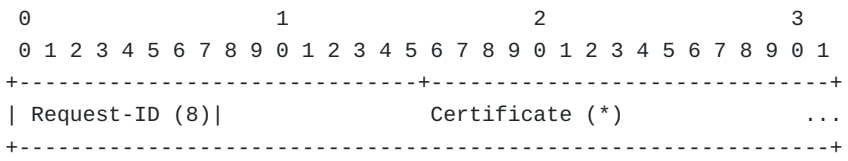

Figure 5: CERTIFICATE frame payload

The fields defined by the "CERTIFICATE" frame are:

Request-ID: The ID of the "CERTIFICATE_REQUEST" to which this frame responds.

Certificate: An X.509v3 [RFC5280] certificate in the sender's certificate chain.

The first or only "CERTIFICATE" frame with a given Request-ID MUST contain the sender's certificate. Each subsequent certificate SHOULD directly certify the certificate immediately preceding it. A certificate which specifies a trust anchor MAY be omitted, provided that the recipient is known to already possess the relevant certificate. (For example, because it was included in a "CERTIFICATE_REQUEST"'s Certificate-Authorities list.)

The "Request-ID" field MUST contain the same value as the corresponding "CERTIFICATE_REQUEST" frame, and the provided certificate chain MUST conform to the requirements expressed in the "CERTIFICATE_REQUEST" to the best of the client's ability. Specifically:

- If the "CERTIFICATE_REQUEST" contained a non-empty "Certificate-Authorities" element, one of the certificates in the chain SHOULD be signed by one of the listed CAs.

- If the "CERTIFICATE_REQUEST" contained a non-empty "Cert-Extensions" element, the first certificate MUST match with regard to the extension OIDs recognized by the client.

- Each certificate that is not self-signed MUST be signed using a hash/signature algorithm listed in the "Algorithms" element.

If these requirements are not satisfied, the server MAY at its discretion either process the request without client authentication, or respond with a stream error [RFC7540] on any stream where the certificate is used. Section 3 defines certificate-related error codes which might be applicable.

A client cannot provide different certificates in response to the same "CERTIFICATE_REQUEST" for use on different streams. A client that has already sent and proven a certificate, but does not wish to use it on a particular stream SHOULD send an empty "USE_CERTIFICATE" frame, refusing to use that certificate on that stream.

2.5. The CERTIFICATE_PROOF Frame

The "CERTIFICATE_PROOF" frame proves possession of the private key corresponding to an end certificate previously shown in a "CERTIFICATE" frame.

The "CERTIFICATE_PROOF" frame defines one flag:

AUTOMATIC_USE (0x01): Indicates that the certificate can be used automatically on future requests.

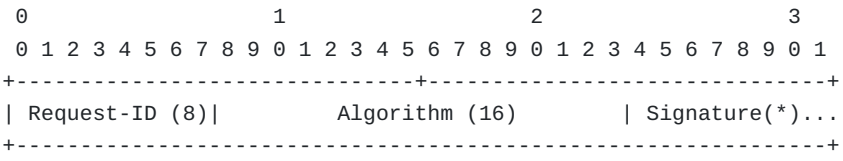

Figure 6: CERTIFICATE_PROOF frame payload

The "CERTIFICATE_PROOF" frame (0xFRAME-TBD4) contains an "Algorithm" field (a "SignatureAndHashAlgorithm", from [I-D.ietf-tls-tls13] section 6.3.2.1), describing the hash/signature algorithm pair being used. The signature is performed as described in [I-D.ietf-tls-tls13], with the following values being used:

- The context string for the signature is "HTTP/2 CERTIFICATE_PROOF"

- The "specified content" is an [RFC5705] exported value, with the following parameters:

  - Disambiguating label string: "EXPORTER HTTP/2 CERTIFICATE_PROOF"

  - Length: 64 bytes

Because the exported value can be independently calculated by both sides of the TLS connection, the value to be signed is not sent on the wire at any time. The same signed value is used for all "CERTIFICATE_PROOF" frames in a single HTTP/2 connection.

A "CERTIFICATE_PROOF" frame MUST be sent only after all "CERTIFICATE" frames with the same Request-ID have been sent, and MUST correspond to the first certificate presented in the first "CERTIFICATE" frame with that Request-ID. Receipt of multiple "CERTIFICATE_PROOF" frames for the same Request-ID, receipt of a "CERTIFICATE_PROOF" frame without a corresponding "CERTIFICATE" frame, or receipt of a "CERTIFICATE" frame after a corresponding "CERTIFICATE_PROOF" MUST be treated as a session error of type "PROTOCOL_ERROR".

If the "AUTOMATIC_USE" flag is set, the server MAY omit sending "CERTIFICATE_REQUIRED" frames on future streams associated with this request and use the referenced certificate for authentication without further notice to the client. This behavior is optional, and receipt of a "CERTIFICATE_REQUIRED" frame does not imply that previously-presented certificates were unacceptable to the server.

Use of the "CERTIFICATE_PROOF" frame by servers is not defined by this document. A "CERTIFICATE_PROOF" frame received by a client MUST be ignored.

3. Indicating failures during HTTP-Layer Certificate Authentication

Because this draft permits client 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

BAD_SIGNATURE (0xERROR-TBD5): The digital signature provided did not match

CERTIFICATE_TOO_LARGE (0xERROR-TBD6): The certificate cannot be transferred due to the recipient's "SETTINGS_MAX_FRAME_SIZE"

CERTIFICATE_GENERAL (0xERROR-TBD7): Any other certificate-related error

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.

4. Indicating Support for HTTP-Layer Certificate Authentication

Clients that support 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 client does not support reactive certificate authentication. A client sets the "SETTINGS_HTTP_CERT_AUTH" setting to a value of 1 to indicate support for HTTP-layer certificate authentication as defined in this document. Any value other than 0 or 1 MUST be treated as a connection error (Section 5.4.1 of [RFC7540]) of type "PROTOCOL_ERROR".

5. Security Considerations

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

In order to protect the privacy of the connection against triple-handshake attacks, this feature of HTTP/2 MUST be used only over TLS

1.3 or greater, or over TLS 1.2 in combination with the Extended Master Secret extension defined in [RFC7627]. Because this feature is intended to operate with equivalent security to the TLS connection, hash and signature algorithms not permitted by the version of TLS in use MUST NOT be used. Additionally, the following algorithms MUST NOT be used, even if permitted by the underlying TLS version:

- MD5
- SHA1
- SHA224
- DSA
- ECDSA with curves on prime fields that are less than 240 bits wide
- RSA with a prime modulus less than 2048 bits

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 do not intend to have associated with the provided certificate.

Implementations need to be aware of the potential for confusion about the state of a connection. The presence or absence of a validated client 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.

6. IANA Considerations

The HTTP/2 "SETTINGS_HTTP_CERT_AUTH" setting is registered in Section 6.1. Five 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].

6.2.1. CERTIFICATE_REQUIRED

Frame Type: CERTIFICATE_REQUIRED

Code: 0xFRAME-TBD1

Specification: This document.

6.2.2. CERTIFICATE_REQUEST

Frame Type: CERTIFICATE_REQUEST

Code: 0xFRAME-TBD2

Specification: This document.

6.2.3. CERTIFICATE

Frame Type: CERTIFICATE

Code: 0xFRAME-TBD3

Specification: This document.

6.2.4. CERTIFICATE_PROOF

Frame Type: CERTIFICATE_PROOF

Code: 0xFRAME-TBD4

Specification: This document.

6.2.5. USE_CERTIFICATE

Frame Type: USE_CERTIFICATE

Code: 0xFRAME-TBD5

Specification: This document.

6.3. New HTTP/2 Error Codes

Five new error codes are registered in the "HTTP/2 Error Code" registry established in [RFC7540].

6.3.1. BAD_CERTIFICATE

Name: BAD_CERTIFICATE

Code: 0xERROR-TBD1

Specification: This document.

6.3.2. UNSUPPORTED_CERTIFICATE

Name: UNSUPPORTED_CERTIFICATE

Code: 0xERROR-TBD2

Specification: This document.

6.3.3. CERTIFICATE_REVOKED

Name: CERTIFICATE_REVOKED

Code: 0xERROR-TBD3

Specification: This document.

6.3.4. CERTIFICATE_EXPIRED

Name: CERTIFICATE_EXPIRED

Code: 0xERROR-TBD4

Specification: This document.

6.3.5. BAD_SIGNATURE

Name: BAD_SIGNATURE

Code: 0xERROR-TBD5

Specification: This document.

6.3.6. CERTIFICATE_GENERAL

Name: CERTIFICATE_GENERAL

Code: 0xERROR-TBD6

Specification: This document.

7. Acknowledgements

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

8. Normative References

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-11 (work in progress), December 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://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, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

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

[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, <http://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, <http://www.rfc-editor.org/info/rfc7540>.

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <http://www.rfc-editor.org/info/rfc7627>.

[X690] ITU-T, "Information technology - ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ISO ISO/IEC 8825-1:2002, 2002, <http://www.itu.int/ITU-T/studygroups/com17/languages/X.690-0207.pdf>.

Authors' Addresses

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

Mike Bishop
Microsoft

Email: michael.bishop@microsoft.com