HTTPbis
Internet-Draft
Intended status: Informational
Expires: May 19, 2017

M. Bishop
Microsoft

November 15, 2016

HTTP/2 Extended SETTINGS Extension
draft-bishop-httpbis-extended-settings-01

Abstract

HTTP/2 defines the SETTINGS frame to contain a single 32-bit value per setting. While this is sufficient to convey everything used in the core HTTP/2 specification, some protocols will require more complex values, such as arrays of code-points or strings.

For such protocols, this extension defines a parallel to the SETTINGS frame, EXTENDED_SETTINGS, where the value of a setting is not a 32-bit value, but a variable-length opaque data blob whose interpretation is subject entirely to the definition of the protocol using it.

Table of Contents

1. Introduction 2
2. Detection of Support 3
3. Extension Frame Types 3
3.1. EXTENDED_SETTINGS Frame 3
3.1.1. EXTENDED_SETTINGS Format 4
3.2. EXTENDED_SETTINGS_ACK Frame 5
4. Settings Synchronization 5
5. Security Considerations 6
6. IANA Considerations 6
6.1. Signature Methods 6
6.2. HTTP/2 SETTINGS_HTTP_CERT_AUTH Setting 6
6.3. New HTTP/2 Frames 7
7. References 7
7.1. Normative References 7
7.2. Informative References 7
Author's Address 8

1. Introduction

In [I-D.bishop-httpbis-http2-additional-certs], values for which IANA registries already exist must be communicated between two HTTP/2 implementations. Since the SETTINGS frame constrains setting values to a 32-bit value, the existing version of that draft divides the 32-bit value into halves and dedicates bits to each currently-known value. This requires the creation of two duplicative IANA registries, and enormously constrains future extensibility since each future supported value will consume one of only sixteen bits. It also causes divergence from other places in the protocol where a bitmask is not required and a more sensible value can be used.

[MS-HTTP2E], likewise, defines a very limited bitmap in the 32-bit value - two bits are defined, all others are reserved (and not useful). The setting fits easily in a single byte, and need not consume a four-byte value every time it is transferred.

Alternately, a number of recent and in-progress HTTP/2 extensions describe properties of the connection that are informative to the peer ([RFC7838], [I-D.ietf-httpbis-origin-frame]). These are essentially settings that did not fit into a 32-bit value.

Each extension could define its own SETTINGS-equivalent frame to carry its own data, as these extensions already have, but to do so every time a new extension might require such a capability seems similarly wasteful, given the limited frame type space (also an IANA registry).

2. Detection of Support

An HTTP/2 peer that supports the EXTENDED_SETTINGS frame indicate this using the HTTP/2 "SETTINGS_EXTENDED_SETTINGS" (0xSETTING-TBD) setting.

The initial value for the "SETTINGS_EXTENDED_SETTINGS" setting is 0 (0x00), indicating that the peer does not support the EXTENDED_SETTINGS frame. A peer that is able to parse the EXTENDED_SETTINGS frame MUST set this value to 1 (0x01).

This setting MUST be sent before any of the frame types in Section 3 are sent, but those frames MAY be sent before the setting is acknowledged and MAY be sent regardless of whether the peer has sent this setting.

3. Extension Frame Types

3.1. EXTENDED_SETTINGS Frame

The EXTENDED_SETTINGS frame (type=0xTBD1) conveys configuration parameters that affect how endpoints communicate, such as preferences and constraints on peer behavior which occur in a form other than a 32-bit value. The EXTENDED_SETTINGS frame is also used to acknowledge the receipt of those parameters. Individually, an EXTENDED_SETTINGS parameter can also be referred to as a "setting".

EXTENDED_SETTINGS parameters are not negotiated; they describe characteristics of the sending peer, which are used by the receiving peer. However, a negotiation can be implied by the use of EXTENDED_SETTINGS - a peer uses EXTENDED_SETTINGS to advertise a set of supported values. The recipient can then choose which entries from this list are also acceptable and proceed with the value it has chosen. (This choice could be announced in a field of an extension frame, or in a value in SETTINGS.)

Different values for the same parameter can be advertised by each peer. For example, a server might support many different signing algorithms, while a resource constrained client has only one or two that it can validate.

An EXTENDED_SETTINGS frame MAY be sent at any time by either endpoint over the lifetime of the connection.

Each parameter in an EXTENDED_SETTINGS frame replaces any existing value for that parameter. Parameters are processed in the order in which they appear, and a receiver of an EXTENDED_SETTINGS frame does not need to maintain any state other than the current value of its parameters. Therefore, the value of a EXTENDED_SETTINGS parameter is the last value that is seen by a receiver.

EXTENDED_SETTINGS parameters can request acknowledgement by the receiving peer. To enable this, the EXTENDED_SETTINGS frame defines the following flag:

REQUEST_ACK (0x1): When set, bit 0 indicates that this frame contains values which the sender wants to know were understood and applied. For more information, see Section 4.

Like SETTINGS frames, EXTENDED_SETTINGS frames always apply to a connection, never a single stream. The stream identifier for an EXTENDED_SETTINGS frame MUST be zero (0x0). If an endpoint receives an EXTENDED_SETTINGS frame whose stream identifier field is anything other than 0x0, the endpoint MUST respond with a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

The EXTENDED_SETTINGS frame affects connection state. A badly formed or incomplete EXTENDED_SETTINGS frame MUST be treated as a connection error (Section 5.4.1) of type PROTOCOL_ERROR.

3.1.1. EXTENDED_SETTINGS Format

The payload of a SETTINGS frame consists of zero or more parameters, each consisting of an unsigned 16-bit setting identifier and a length-prefixed binary value.

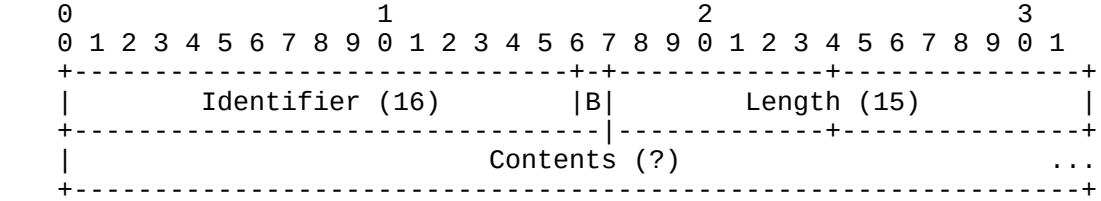


Figure 1: EXTENDED_SETTINGS frame payload

A zero-length content indicates that the setting value is a Boolean given by the B bit. If Length is not zero, the B bit MUST be zero,

and MUST be ignored by receivers. The initial value of each setting is "false."

An implementation MUST ignore the contents for any EXTENDED_SETTINGS identifier it does not understand.

3.2. EXTENDED_SETTINGS_ACK Frame

The EXTENDED_SETTINGS_ACK frame acknowledges receipt and application of specific values in the peer's SETTINGS frame. It contains a list of EXTENDED_SETTINGS identifiers which the sender has understood and applied. This list MAY be empty.

Any EXTENDED_SETTINGS_ACK frame whose length is not a multiple of two bytes MUST be treated as a connection error ([RFC7540] section 5.4.1) of type "FRAME_SIZE_ERROR".

4. Settings Synchronization

Some values in EXTENDED_SETTINGS benefit from or require an understanding of when the peer has received and applied the changed parameter values. In order to provide such synchronization timepoints, the recipient of a EXTENDED_SETTINGS frame MUST apply the updated parameters as soon as possible upon receipt. The values in the EXTENDED_SETTINGS frame MUST be processed in the order they appear, with no other frame processing between values. Unsupported parameters MUST be ignored.

Once all values have been processed, if the REQUEST_ACK flag was set, the recipient MUST immediately emit a EXTENDED_SETTINGS_ACK frame listing the identifiers whose values were understood and applied. (If none of the values were understood, the EXTENDED_SETTINGS_ACK frame will be empty, but MUST still be sent.) Upon receiving an EXTENDED_SETTINGS_ACK frame, the sender of the altered parameters can rely on the setting having been applied.

If the sender of an EXTENDED_SETTINGS frame with the "REQUEST_ACK" flag set does not receive an acknowledgement from a peer that has sent the "SETTINGS_EXTENDED_SETTINGS" setting within a reasonable amount of time, it MAY issue a connection error ([RFC7540] Section 5.4.1) of type SETTINGS_TIMEOUT. This error MUST NOT be sent if the peer has not previously advertised support for EXTENDED_SETTINGS.

5. Security Considerations

Because these frames can be used to request that peers retain potentially-large state, implementations need to use caution in their retention policies. Values which are not understood MUST be discarded in order to protect against increased memory usage. Specifications which make use of EXTENDED_SETTINGS MUST include details about how the contents can be parsed and stored, and SHOULD include details about how the information can be compressed and when it can safely be discarded.

6. IANA Considerations

This draft establishes one new registry and add three entries across two existing registries.

The HTTP/2 "SETTINGS_EXTENDED_SETTINGS" setting is registered in Section 6.2. Two frame types are registered in Section 6.3.

6.1. Signature Methods

This document establishes a registry for HTTP/2 extended settings. The "HTTP/2 Extended Settings" registry manages a 16-bit space. The "HTTP/2 Extended Settings" registry operates under the "Expert Review" policy [RFC5226] for values in the range from 0x0000 to 0xefff, with values between and 0xf000 and 0xffff being reserved for Experimental Use.

New registrations are advised to provide the following information:

Name: A symbolic name for the setting. Specifying a setting name is optional.

Code: The 16-bit code assigned to the setting.

Specification: An optional reference to a specification that describes the use of the setting.

No entries are registered by this document.

6.2. HTTP/2 SETTINGS_HTTP_CERT_AUTH Setting

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

Name: SETTINGS_EXTENDED_SETTINGS

Code: 0xSETTING-TBD

Initial Value: 0

Specification: This document.

6.3. New HTTP/2 Frames

Two 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
EXTENDED_SETTINGS	0xFRAME-TBD1	{{settings-frame}}
EXTENDED_SETTINGS_ACK	0xFRAME-TBD2	{{ack}}

Figure 2

7. References

7.1. Normative References

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[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>.

7.2. Informative References

[I-D.bishop-httpbis-http2-additional-certs] Bishop, M. and M. Thomson, "Secondary Certificate Authentication in HTTP/2", draft-bishop-httpbis-http2-additional-certs-02 (work in progress), October 2016.

[I-D.ietf-httpbis-origin-frame] Nottingham, M. and E. Nygren, "The ORIGIN HTTP/2 Frame", draft-ietf-httpbis-origin-frame-01 (work in progress), September 2016.

[MS-HTTP2E]
"Hypertext Transfer Protocol Version 2 (HTTP/2) Extension", October 2015, <http://download.microsoft.com/download/9/5/E/95EF66AF-9026-4BB0-A41D-A4F81802D92C/[MS-HTTP2E].pdf>.

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

Author's Address

Mike Bishop
Microsoft

Email: michael.bishop@microsoft.com

HTTP
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2018

M. Bishop

N. Sullivan
Cloudflare
M. Thomson
Mozilla

October 30, 2017

Secondary Certificate Authentication in HTTP/2
draft-bishop-httpbis-http2-additional-certs-05

Abstract

TLS provides fundamental mutual authentication services for HTTP, supporting up to one server certificate and up to one client certificate associated to the session to prove client and server identities as necessary. This draft provides mechanisms for providing additional such certificates at the HTTP layer when these constraints are not sufficient.

Many HTTP servers host content from several origins. HTTP/2 [RFC7540] 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 [I-D.ietf-tls-tls13] handshake.

In many cases, servers will wish to maintain separate certificates for different origins but still desire the benefits of a shared HTTP connection. Similarly, servers may require clients to present authentication, but have different requirements based on the content the client is attempting to access.

This document describes how TLS exported authenticators [I-D.ietf-tls-exported-authenticator] can be used to provide proof of ownership of additional certificates to the HTTP layer to support both scenarios.

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/.

Copyright Notice

Table of Contents

1. Introduction 3
  1.1. Server Certificate Authentication 3
  1.2. Client Certificate Authentication 4
    1.2.1. HTTP/1.1 using TLS 1.2 and previous 5
    1.2.2. HTTP/1.1 using TLS 1.3 6
    1.2.3. HTTP/2 6
  1.3. HTTP-Layer Certificate Authentication 7
  1.4. Terminology 8
2. Discovering Additional Certificates at the HTTP/2 Layer . . . 8
  2.1. Indicating support for HTTP-layer certificate authentication 8
  2.2. Making certificates or requests available 8
  2.3. Requiring certificate authentication 9
3. Certificates Frames for HTTP/2 11
  3.1. The CERTIFICATE_NEEDED frame 11
  3.2. The USE_CERTIFICATE Frame 12
  3.3. The CERTIFICATE_REQUEST Frame 13
  3.4. The CERTIFICATE Frame 14
    3.4.1. Exported Authenticator Characteristics 15
4. Indicating failures during HTTP-Layer Certificate Authentication 15
5. Security Considerations 16
  5.1. Impersonation 16
  5.2. Fingerprinting 17

5.3. Denial of Service 17
5.4. Confusion About State 17
6. IANA Considerations 18
6.1. HTTP/2 SETTINGS_HTTP_CERT_AUTH Setting 18
6.2. New HTTP/2 Frames 18
6.3. New HTTP/2 Error Codes 19
7. Acknowledgements 19
8. References 19
8.1. Normative References 19
8.2. Informative References 21
Authors' Addresses 21

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 X.509 certificates 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.

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.

[I-D.ietf-httpbis-origin-frame] 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 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.

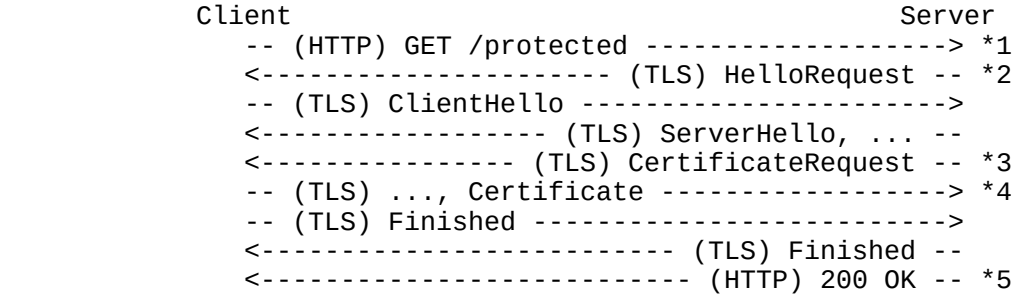


Figure 1: HTTP/1.1 Reactive Certificate Authentication with TLS 1.2

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).

1.2.2. 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.

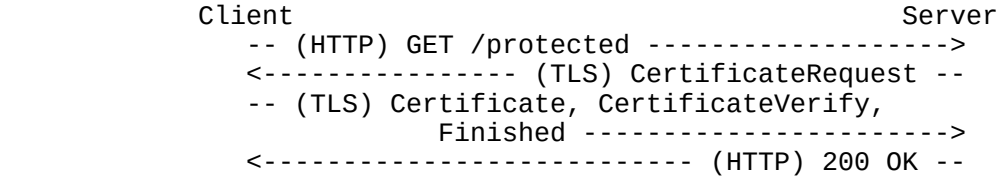


Figure 2: HTTP/1.1 Reactive Certificate Authentication with TLS 1.3

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 the on-stream frames incorporate them 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. An authenticator message can be constructed by either the client or the server given an established TLS connection, a certificate, and a corresponding private key. Exported Authenticators use the message structures from section 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. These pre-supplied elements are then available for automatic use (in some situations) or for reference by individual streams.

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

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

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 single "CERTIFICATE" frame (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. If the certificate is marked as "AUTOMATIC_USE", the certificate may be used by the recipient to authorize any current or future request. Otherwise, the recipient requests the required certificate on each stream, but the previously-supplied certificates are available for reference without having to resend them.

Likewise, the details of a request are sent on stream zero and stored by the recipient. These details will be referenced by subsequent "CERTIFICATE_NEEDED" frames.

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.

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 1.

2.2. Making certificates or requests available

When a peer has advertised support for HTTP-layer certificates as in Section 2.1, either party can supply additional certificates into the connection at any time. These certificates then become available for the peer to consider when deciding whether a connection is suitable to transport a particular request.

Available certificates which have the "AUTOMATIC_USE" flag set MAY be used by the recipient without further notice. This means that clients or servers which predict a certificate will be required could

pre-supply the certificate without being asked. Regardless of whether "AUTOMATIC_USE" is set, these certificates are available for reference by future "USE_CERTIFICATE" frames.

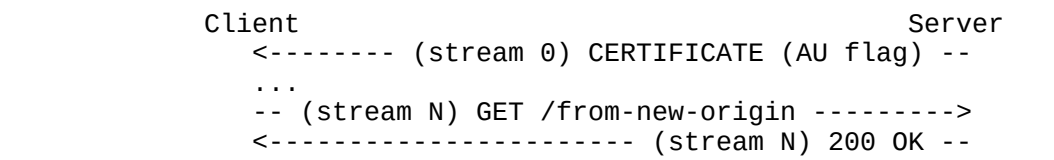


Figure 3: Proactive Server Certificate

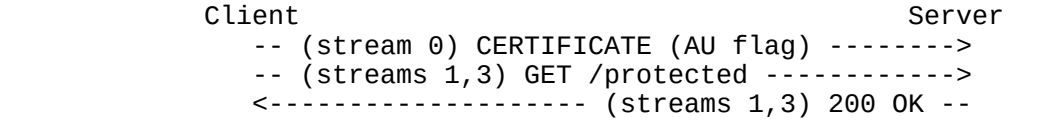


Figure 4: Proactive Client Certificate

Likewise, either party can supply a "CERTIFICATE_REQUEST" that outlines parameters of a certificate they might request in the future. It is important to note that this does not currently request such a certificate, but makes the contents of the request available for reference by a future "CERTIFICATE_NEEDED" frame.

2.3. Requiring certificate authentication

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 [I-D.ietf-httpbis-origin-frame]) and advertised support for HTTP-layer certificates (see Section 2.1), it MAY send a "CERTIFICATE_NEEDED" frame on the stream it will use to make the request. (If the request parameters have not already been made available using a "CERTIFICATE_REQUEST" frame, the client will need to send the "CERTIFICATE_REQUEST" in order to generate the "CERTIFICATE_NEEDED" frame.) The stream represents a pending request to that origin which is blocked until a valid certificate is processed.

The request is blocked until the server has responded with a "USE_CERTIFICATE" frame pointing to a certificate for that origin. If the certificate is already available, the server SHOULD immediately respond with the appropriate "USE_CERTIFICATE" frame. (If the certificate has not already been transmitted, the server will need to make the certificate available as described in Section 2.2 before completing the exchange.)

If the server does not have the desired certificate, it MUST respond with an empty "USE_CERTIFICATE" frame. 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.

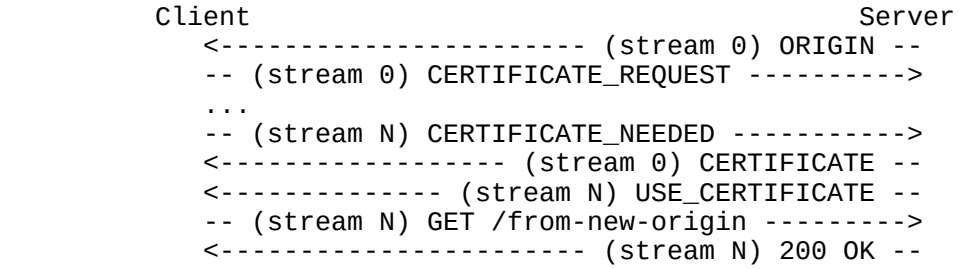


Figure 5: Client-Requested Certificate

Likewise, on each stream where certificate authentication is required, the server sends a "CERTIFICATE_NEEDED" frame, which the client answers with a "USE_CERTIFICATE" frame indicating the certificate to use. 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.

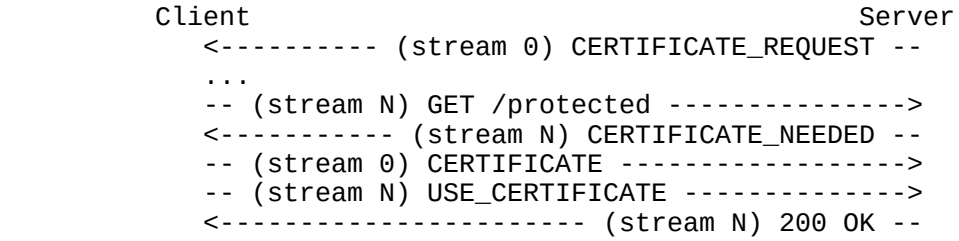


Figure 6: Reactive Certificate Authentication

A server SHOULD provide certificates for an origin before pushing resources from it or supplying content referencing the origin. If a

client receives a "PUSH_PROMISE" referencing an origin for which it has not yet received the server's certificate, the client MUST verify the server's possession of an appropriate certificate by sending a "CERTIFICATE_NEEDED" frame on the pushed stream to inform the server that progress is blocked until the request is satisfied. The client MUST NOT use the pushed resource until an appropriate certificate has been received and validated.

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 on 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.

"Request-ID" and "Cert-ID" are sender-local, and the use of the same value by the other peer does not imply any correlation between their frames. These values MUST be unique per sender over the lifetime of the connection.

3.1. The CERTIFICATE_NEEDED frame

The "CERTIFICATE_NEEDED" frame (0xFRAME-TBD1) is sent to indicate that the HTTP request on the current stream 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 sent on stream zero. The "CERTIFICATE_REQUEST" describes the certificate the sender requires to make progress on the stream in question.

The "CERTIFICATE_NEEDED" frame contains 2 octets, which is 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 on 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 on the same stream at any given time.

Clients MUST NOT send multiple "CERTIFICATE_NEEDED" frames on the same stream.

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 be sent on stream zero, and MUST NOT be sent on a stream in the "half-closed (local)" state [RFC7540]. A client that receives a "CERTIFICATE_NEEDED" frame on 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 in response to a "CERTIFICATE_NEEDED" frame to indicate which certificate is being used to satisfy the requirement.

A "USE_CERTIFICATE" frame with no payload 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.

Otherwise, the "USE_CERTIFICATE" frame contains 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.

The "USE_CERTIFICATE" frame MUST NOT be sent on stream zero or a stream on which a "CERTIFICATE_NEEDED" frame has not been received. Receipt of a "USE_CERTIFICATE" frame in these circumstances SHOULD be treated as a stream error of type "PROTOCOL_ERROR". Each "USE_CERTIFICATE" frame should reference a preceding "CERTIFICATE" frame. Receipt of a "USE_CERTIFICATE" frame before the necessary frames have been received on stream zero MUST also result in a stream error of type "PROTOCOL_ERROR".

The referenced certificate chain MUST conform to the requirements expressed in the "CERTIFICATE_REQUEST" to the best of the sender’s

ability. Specifically, if the "CERTIFICATE_REQUEST" contained a non-empty "Cert-Extensions" element, the end-entity certificate MUST match with regard to the extensions recognized by the sender.

If these requirements are not satisfied, the recipient 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

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-TBD2), which uses the same set of extensions to specify a desired certificate, but can be sent over any TLS version and can be sent by either peer.

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".

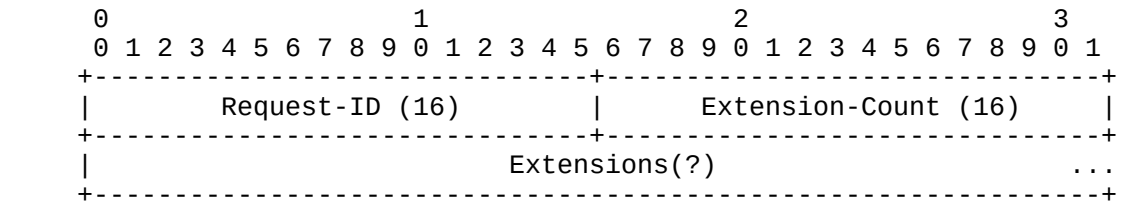


Figure 7: CERTIFICATE_REQUEST frame payload

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.

Extension-Count and Extensions: A list of certificate selection criteria, represented in a series of "Extension" structures (see [I-D.ietf-tls-tls13] section 4.2). This criteria MUST be used in certificate selection as described in [I-D.ietf-tls-tls13]. The number of "Extension" structures is given by the 16-bit "Extension-Count" field, which MAY be zero.

Some extensions used for certificate selection allow multiple values (e.g. oid_filters on Extended Key Usage). If the sender has included a non-empty Extensions list, the certificate MUST match all criteria specified by extensions the recipient recognizes. However, the recipient MUST ignore and skip any unrecognized certificate selection extensions.

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:

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

TO_BE_CONTINUED (0x02): Indicates that the exported authenticator spans more than one frame.

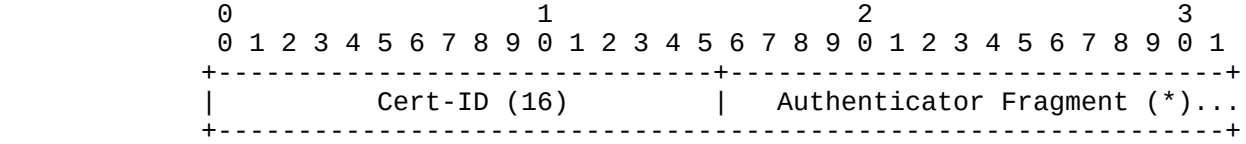


Figure 8: CERTIFICATE frame payload

The "Exported Authenticator Fragment" field contains 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.

This opaque data 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".

If the "AUTOMATIC_USE" flag is set, the recipient MAY omit sending "CERTIFICATE_NEEDED" frames on future streams which would require a similar certificate and use the referenced certificate for authentication without further notice to the holder. This behavior is optional, and receipt of a "CERTIFICATE_NEEDED" frame does not imply that previously-presented certificates were unacceptable, even if "AUTOMATIC_USE" was set. Servers MUST set the "AUTOMATIC_USE" flag when sending a "CERTIFICATE" frame. A server MUST NOT send certificates for origins which it is not prepared to service on the current connection.

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 certificate, supporting information about the certificate (OCSP, SCT, etc.), and an optional "certificate_request_context". When generating exported authenticators for use with this extension, the "certificate_request_context" MUST be the two-octet Cert-ID.

Upon receipt of a completed authenticator, an endpoint MUST check that:

- the "validate" API confirms the validity of the authenticator itself
- the "certificate_request_context" matches the Cert-ID of the frame(s) in which it was received

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

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 [I-D.ietf-httpbis-origin-frame], 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 colocated 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

Five 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

7. Acknowledgements

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

8. References

8.1. Normative References

[I-D.ietf-tls-exported-authenticator]
Sullivan, N., "Exported Authenticators in TLS", draft-ietf-tls-exported-authenticator-03 (work in progress), July 2017.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-21 (work in progress), July 2017.

[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>.

[RFC2459] Housley, R., Ford, W., Polk, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 2459, DOI 10.17487/RFC2459, January 1999, <https://www.rfc-editor.org/info/rfc2459>.

[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>.

[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, <https://www.rfc-editor.org/info/rfc5280>.

[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>.

[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>.

8.2. Informative References

[I-D.ietf-httpbis-origin-frame]
Nottingham, M. and E. Nygren, "The ORIGIN HTTP/2 Frame", draft-ietf-httpbis-origin-frame-04 (work in progress), August 2017.

[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>.

Authors' Addresses

Mike Bishop

Email: mbishop@evequefou.be

Nick Sullivan
Cloudflare

Email: nick@cloudflare.com

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

HTTP Working Group
Internet-Draft
Intended status: Experimental
Expires: January 3, 2019

K. Oku
Fastly
Y. Weiss
Akamai

July 2, 2018

Cache Digests for HTTP/2
draft-ietf-httpbis-cache-digest-05

Abstract

This specification defines a HTTP/2 frame type to allow clients to inform the server of their cache's contents. Servers can then use this to inform their choices of what to push to clients.

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/ .

Working Group information can be found at http://httpwg.github.io/ ; source code and issues list for this draft can be found at https://github.com/httpwg/http-extensions/labels/cache-digest .

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 January 3, 2019.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Notational Conventions 3
2. The CACHE_DIGEST Frame 3
2.1. Client Behavior 4
2.1.1. Creating a digest 4
2.1.2. Adding a URL to the Digest-Value 5
2.1.3. Removing a URL to the Digest-Value 7
2.1.4. Computing a fingerprint value 8
2.1.5. Computing the key 9
2.1.6. Computing a Hash Value 9
2.1.7. Computing an Alternative Hash Value 9
2.2. Server Behavior 10
2.2.1. Querying the Digest for a Value 10
3. The SETTINGS_SENDING_CACHE_DIGEST SETTINGS Parameter 11
4. The SETTINGS_ACCEPT_CACHE_DIGEST SETTINGS Parameter 12
5. IANA Considerations 12
6. Security Considerations 13
7. References 13
7.1. Normative References 13
7.2. Informative References 14
Appendix A. Encoding the CACHE_DIGEST frame as an HTTP Header . 15
Appendix B. Changes 16
B.1. Since draft-ietf-httpbis-cache-digest-04 16
B.2. Since draft-ietf-httpbis-cache-digest-03 16
B.3. Since draft-ietf-httpbis-cache-digest-02 16
B.4. Since draft-ietf-httpbis-cache-digest-01 16
B.5. Since draft-ietf-httpbis-cache-digest-00 17
Appendix C. Acknowledgements 17
Authors' Addresses 17

1. Introduction

HTTP/2 [RFC7540] allows a server to "push" synthetic request/response pairs into a client's cache optimistically. While there is strong interest in using this facility to improve perceived Web browsing

performance, it is sometimes counterproductive because the client might already have cached the "pushed" response.

When this is the case, the bandwidth used to "push" the response is effectively wasted, and represents opportunity cost, because it could be used by other, more relevant responses. HTTP/2 allows a stream to be cancelled by a client using a RST_STREAM frame in this situation, but there is still at least one round trip of potentially wasted capacity even then.

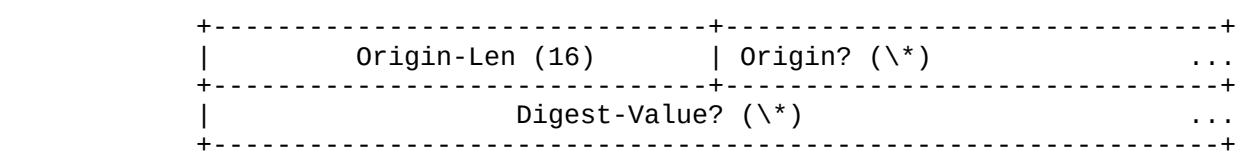
This specification defines a HTTP/2 frame type to allow clients to inform the server of their freshly cached contents using a Cuckoo-filter [Cuckoo] based digest. Servers can then use this to inform their choices of what to push to clients.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. The CACHE_DIGEST Frame

The CACHE_DIGEST frame type is 0xd (decimal 13).



The CACHE_DIGEST frame payload has the following fields:

Origin-Len: An unsigned, 16-bit integer indicating the length, in octets, of the Origin field.

Origin: A sequence of characters containing the ASCII serialization of an origin ([RFC6454], Section 6.2) that the Digest-Value applies to.

Digest-Value: A sequence of octets containing the digest as computed in Section 2.1.1 and Section 2.1.2.

The CACHE_DIGEST frame defines the following flags:

- *RESET* (0x1): When set, indicates that any and all cache digests for the applicable origin held by the recipient MUST be considered invalid.

- *COMPLETE* (0x2): When set, indicates that the currently valid set of cache digests held by the server constitutes a complete representation of the cache's state regarding that origin.

2.1. Client Behavior

A CACHE_DIGEST frame MUST be sent from a client to a server on stream 0, and conveys a digest of the contents of the client's cache for the indicated origin.

In typical use, a client will send one or more CACHE_DIGESTs immediately after the first request on a connection for a given origin, on the same stream, because there is usually a short period of inactivity then, and servers can benefit most when they understand the state of the cache before they begin pushing associated assets (e.g., CSS, JavaScript and images). Clients MAY send CACHE_DIGEST at other times.

If the cache's state is cleared, lost, or the client otherwise wishes the server to stop using previously sent CACHE_DIGESTs, it can send a CACHE_DIGEST with the RESET flag set.

When generating CACHE_DIGEST, a client MUST NOT include stale-cached responses or responses whose URLs do not share origins [RFC6454] with the indicated origin. Clients MUST NOT send CACHE_DIGEST frames on connections that are not authoritative (as defined in [RFC7540], 10.1) for the indicated origin.

When the CACHE_DIGEST frames sent represent the complete set of stored responses, the last such frame SHOULD have a COMPLETE flag set, to indicate to the server that it has all relevant state. Note that for the purposes of COMPLETE, responses cached since the beginning of the connection or the last RESET flag on a CACHE_DIGEST frame need not be included.

CACHE_DIGEST has no defined meaning when sent from servers, and SHOULD be ignored by clients.

2.1.1. Creating a digest

Given the following inputs:

- "P", an integer smaller than 256, that indicates the probability of a false positive that is acceptable, expressed as "1/2**P".

- "N", an integer that represents the number of entries - a prime number smaller than 2**32

1. Let "f" be the number of bits per fingerprint, calculated as "P + 3"

2. Let "b" be the bucket size, defined as 4.

3. Let "allocated" be the closest power of 2 that is larger than "N".

4. Let "bytes" be "f"*"allocated"*"b"/8 rounded up to the nearest integer

5. Add 5 to "bytes"

6. Allocate memory of "bytes" and set it to zero. Assign it to "digest-value".

7. Set the first byte to "P"

8. Set the second till fifth bytes to "N" in big endian form

9. Return the "digest-value".

Note: "allocated" is necessary due to the nature of the way Cuckoo filters are creating the secondary hash, by XORing the initial hash and the fingerprint's hash. The XOR operation means that secondary hash can pick an entry beyond the initial number of entries, up to the next power of 2. In order to avoid issues there, we allocate the table appropriately. For increased space efficiency, it is recommended that implementations pick a number of entries that's close to the next power of 2.

2.1.2. Adding a URL to the Digest-Value

Given the following inputs:

- "URL" a string corresponding to the Effective Request URI ([RFC7230], Section 5.5) of a cached response [RFC7234]

- "maxcount" - max number of cuckoo hops

- "digest-value"

1. Let "f" be the value of the first byte of "digest-value".

2. Let "b" be the bucket size, defined as 4.

3. Let "N" be the value of the second to fifth bytes of "digest-value" in big endian form.

4. Let "key" be the return value of Section 2.1.5 with "URL" as input.

5. Let "h1" be the return value of Section 2.1.6 with "key" and "N" as inputs.

6. Let "dest_fingerprint" be the return value of Section 2.1.4 with "key" and "f" as inputs.

7. Let "h2" be the return value of Section 2.1.7 with "h1", "dest_fingerprint" and "N" as inputs.

8. Let "h" be either "h1" or "h2", picked in random.

9. While "maxcount" is larger than zero:

    1. Let "position_start" be 40 + "h" * "f" * "b".

    2. Let "position_end" be "position_start" + "f" * "b".

    3. While "position_start" < "position_end":

        1. Let "bits" be "f" bits from "digest_value" starting at "position_start".

        2. If "bits" is all zeros, set "bits" to "dest_fingerprint" and terminate these steps.

        3. Add "f" to "position_start".

    4. Let "e" be a random number from 0 to "b".

    5. Subtract "f" * ("b" - "e") from "position_start".

    6. Let "bits" be "f" bits from "digest_value" starting at "position_start".

    7. Let "fingerprint" be the value of bits, read as big endian.

    8. Set "bits" to "dest_fingerprint".

    9. Set "dest_fingerprint" to "fingerprint".

    10. Let "h" be Section 2.1.7 with "h", "dest_fingerprint" and "N" as inputs.

    11. Subtract 1 from "maxcount".

10. Subtract "f" from "position_start".

11. Let "fingerprint" be the "f" bits starting at "position_start".

12. Let "h1" be "h"

13. Subtract 1 from "maxcount".

14. If "maxcount" is zero, return an error.

15. Go to step 7.

2.1.3. Removing a URL to the Digest-Value

Given the following inputs:

- "URL" a string corresponding to the Effective Request URI ([RFC7230], Section 5.5) of a cached response [RFC7234]
- "digest-value"

1. Let "f" be the value of the first byte of "digest-value".
2. Let "b" be the bucket size, defined as 4.
3. Let "N" be the value of the second to fifth bytes of "digest-value" in big endian form.
4. Let "key" be the return value of Section 2.1.5 with "URL" as input.
5. Let "h1" be the return value of Section 2.1.6 with "key" and "N" as inputs.
6. Let "fingerprint" be the return value of Section 2.1.4 with "key" and "f" as inputs.
7. Let "h2" be the return value of Section 2.1.7 with "h1", "fingerprint" and "N" as inputs.
8. Let "hashes" be an array containing "h1" and "h2".
9. For each "h" in "hashes":
    1. Let "position_start" be 40 + "h" * "f" * "b".
    2. Let "position_end" be "position_start" + "f" * "b".

3. While "position_start" < "position_end":

    1. Let "bits" be "f" bits from "digest_value" starting at "position_start".

    2. If "bits" is "fingerprint", set "bits" to all zeros and terminate these steps.

    3. Add "f" to "position_start".

2.1.4. Computing a fingerprint value

Given the following inputs:

- "key", an array of characters
- "f", an integer indicating the number of output bits

1. Let "hash-value" be the SHA-256 message digest [RFC6234] of "key", expressed as an integer.
2. Let "h" be the number of bits in "hash-value"
3. Let "fingerprint-value" be 0
4. While "fingerprint-value" is 0 and "h" > "f":
    1. Let "fingerprint-value" be the "f" least significant bits of "hash-value".
    2. Let "hash-value" be the "h"-"f" most significant bits of "hash-value".
    3. Subtract "f" from "h".
5. If "fingerprint-value" is 0, let "fingerprint-value" be 1.
6. Return "fingerprint-value".

Note: Step 5 is to handle the extremely unlikely case where a SHA-256 digest of "key" is all zeros. The implications of it means that there's an infitisimaly larger probability of getting a "fingerprint-value" of 1 compared to all other values. This is not a problem for any practical purpose.

2.1.5. Computing the key

Given the following inputs:

- "URL", an array of characters

1. Let "key" be "URL" converted to an ASCII string by percent-encoding as appropriate [RFC3986].

2. Return "key"

2.1.6. Computing a Hash Value

Given the following inputs:

- "key", an array of characters.

- "N", an integer

"hash-value" can be computed using the following algorithm:

1. Let "hash-value" be the SHA-256 message digest [RFC6234] of "key", truncated to 32 bits, expressed as an integer.

2. Return "hash-value" modulo N.

2.1.7. Computing an Alternative Hash Value

Given the following inputs:

- "hash1", an integer indicating the previous hash.

- "fingerprint", an integer indicating the fingerprint value.

- "N", an integer indicating the number of entries in the digest.

1. Let "fingerprint-string" be the value of "fingerprint" in base 10, expressed as a string.

2. Let "hash2" be the return value of Section 2.1.6 with "fingerprint-string" and "N" as inputs, XORed with "hash1".

3. Return "hash2".

2.2. Server Behavior

In typical use, a server will query (as per Section 2.2.1) the CACHE_DIGESTs received on a given connection to inform what it pushes to that client;

- If a given URL has a match in a current CACHE_DIGEST, a complete response need not be pushed; The server MAY push a 304 response for that resource, indicating the client that it hasn't changed.

- If a given URL has no match in any current CACHE_DIGEST, the client does not have a cached copy, and a complete response can be pushed.

Servers MAY use all CACHE_DIGESTs received for a given origin as current, as long as they do not have the RESET flag set; a CACHE_DIGEST frame with the RESET flag set MUST clear any previously stored CACHE_DIGESTs for its origin. Servers MUST treat an empty Digest-Value with a RESET flag set as effectively clearing all stored digests for that origin.

Clients are not likely to send updates to CACHE_DIGEST over the lifetime of a connection; it is expected that servers will separately track what cacheable responses have been sent previously on the same connection, using that knowledge in conjunction with that provided by CACHE_DIGEST.

Servers MUST ignore CACHE_DIGEST frames sent on a stream other than 0.

2.2.1. Querying the Digest for a Value

Given the following inputs:

- "URL" a string corresponding to the Effective Request URI ([RFC7230], Section 5.5) of a cached response [RFC7234].

- "digest-value", an array of bits.

1. Let "f" be the value of the first byte of "digest-value".

2. Let "b" be the bucket size, defined as 4.

3. Let "N" be the value of the second to fifth bytes of "digest-value" in big endian form.

4. Let "key" be the return value of Section 2.1.5 with "URL" as input.

5. Let "h1" be the return value of Section 2.1.6 with "key" and "N" as inputs.

6. Let "fingerprint" be the return value of Section 2.1.4 with "key" and "f" as inputs.

7. Let "h2" be the return value of Section 2.1.7 with "h1", "fingerprint" and "N" as inputs.

8. Let "hashes" be an array containing "h1" and "h2".

9. For each "h" in "hashes":

    1. Let "position_start" be 40 + "h" * "f" * "b".

    2. Let "position_end" be "position_start" + "f" * "b".

    3. While "position_start" < "position_end":

        1. Let "bits" be "f" bits from "digest_value" starting at "position_start".

        2. If "bits" is "fingerprint", return true

        3. Add "f" to "position_start".

10. Return false.

3. The SETTINGS_SENDING_CACHE_DIGEST SETTINGS Parameter

A Client SHOULD notify its support for CACHE_DIGEST frames by sending the SETTINGS_SENDING_CACHE_DIGEST (0xXXX) SETTINGS parameter.

The value of the parameter is a bit-field of which the following bits are defined:

DIGEST_PENDING (0x1): When set it indicates that the client has a digest to send, and the server may choose to wait for a digest in order to make server push decisions.

Rest of the bits MUST be ignored and MUST be left unset when sending.

The initial value of the parameter is zero (0x0) meaning that the client has no digest to send the server.

4. The SETTINGS_ACCEPT_CACHE_DIGEST SETTINGS Parameter

A server can notify its support for CACHE_DIGEST frame by sending the SETTINGS_ACCEPT_CACHE_DIGEST (0x7) SETTINGS parameter. If the server is tempted to making optimizations based on CACHE_DIGEST frames, it SHOULD send the SETTINGS parameter immediately after the connection is established.

The value of the parameter is a bit-field of which the following bits are defined:

ACCEPT (0x1): When set, it indicates that the server is willing to make use of a digest of cached responses.

Rest of the bits MUST be ignored and MUST be left unset when sending.

The initial value of the parameter is zero (0x0) meaning that the server is not interested in seeing a CACHE_DIGEST frame.

Some underlying transports allow the server’s first flight of application data to reach the client at around the same time when the client sends it’s first flight data. When such transport (e.g., TLS 1.3 [I-D.ietf-tls-tls13] in full-handshake mode) is used, a client can postpone sending the CACHE_DIGEST frame until it receives a SETTINGS_ACCEPT_CACHE_DIGEST settings value.

When the underlying transport does not have such property (e.g., TLS 1.3 in 0-RTT mode), a client can reuse the settings value found in previous connections to that origin [RFC6454] to make assumptions.

5. IANA Considerations

This document registers the following entry in the Permanent Message Headers Registry, as per [RFC3864]:

- Header field name: Cache-Digest
- Applicable protocol: http
- Status: experimental
- Author/Change controller: IESG
- Specification document(s): [this document]

This document registers the following entry in the HTTP/2 Frame Type Registry, as per [RFC7540]:

- Frame Type: CACHE_DIGEST
- Code: 0xd
- Specification: [this document]

This document registers the following entry in the HTTP/2 Settings Registry, as per [RFC7540]:

- Code: 0x7
- Name: SETTINGS_ACCEPT_CACHE_DIGEST
- Initial Value: 0x0
- Reference: [this document]

6. Security Considerations

The contents of a User Agent's cache can be used to re-identify or "fingerprint" the user over time, even when other identifiers (e.g., Cookies [RFC6265]) are cleared.

CACHE_DIGEST allows such cache-based fingerprinting to become passive, since it allows the server to discover the state of the client's cache without any visible change in server behaviour.

As a result, clients MUST mitigate for this threat when the user attempts to remove identifiers (e.g., "clearing cookies"). This could be achieved in a number of ways; for example: by clearing the cache, by changing one or both of N and P, or by adding new, synthetic entries to the digest to change its contents.

TODO: discuss how effective the suggested mitigations actually would be.

Additionally, User Agents SHOULD NOT send CACHE_DIGEST when in "privacy mode."

7. References

7.1. Normative References

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <https://www.rfc-editor.org/info/rfc6454>.

[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>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <https://www.rfc-editor.org/info/rfc7232>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <https://www.rfc-editor.org/info/rfc7234>.

[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>.

7.2. Informative References

[Cuckoo] "Cuckoo Filter: Practically Better Than Bloom", n.d., <https://www.cs.cmu.edu/~dga/papers/cuckoo-conext2014.pdf>.

[Fetch] "Fetch Standard", n.d., <https://fetch.spec.whatwg.org/>.

[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.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/info/rfc3864>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[Service-Workers] Russell, A., Song, J., Archibald, J., and M. Kruisselbrink, "Service Workers 1", W3C Working Draft WD-service-workers-1-20161011, October 2016, <https://www.w3.org/TR/2016/WD-service-workers-1-20161011/>.

Appendix A. Encoding the CACHE_DIGEST frame as an HTTP Header

On some web browsers that support Service Workers [Service-Workers] but not Cache Digests (yet), it is possible to achieve the benefit of using Cache Digests by emulating the frame using HTTP Headers.

For the sake of interoperability with such clients, this appendix defines how a CACHE_DIGEST frame can be encoded as an HTTP header named "Cache-Digest".

The definition uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with the list rule extension defined in [RFC7230], Section 7.

```
Cache-Digest  = 1#digest-entity
digest-entity = digest-value *(OWS ";" OWS digest-flag)
digest-value  = <Digest-Value encoded using base64url>
digest-flag   = token
```

A Cache-Digest request header is defined as a list construct of cache-digest-entities. Each cache-digest-entity corresponds to a CACHE_DIGEST frame.

Digest-Value is encoded using base64url [RFC4648], Section 5. Flags that are set are encoded as digest-flags by their names that are compared case-insensitively.

Origin is omitted in the header form. The value is implied from the value of the ":authority" pseudo header. Client MUST only send Cache-Digest headers containing digests that belong to the origin specified by the HTTP request.

The example below contains a digest of one resource and has only the "COMPLETE" flag set.

```
Cache-Digest: AfdA; complete
```

Clients MUST associate Cache-Digest headers to every HTTP request, since Fetch [Fetch] - the HTTP API supported by Service Workers - does not define the order in which the issued requests will be sent to the server nor guarantees that all the requests will be transmitted using a single HTTP/2 connection.

Also, due to the fact that any header that is supplied to Fetch is required to be end-to-end, there is an ambiguity in what a Cache-Digest header respresents when a request is transmitted through a proxy. The header may represent the cache state of a client or that of a proxy, depending on how the proxy handles the header.

Appendix B. Changes

B.1. Since draft-ietf-httpbis-cache-digest-04

- Remove ETag from the digest key calculations.
- Add SETTINGS_ prefix to parameter names.

B.2. Since draft-ietf-httpbis-cache-digest-03

- Yoav becomes an author; Mark steps down.

B.3. Since draft-ietf-httpbis-cache-digest-02

- Switch to Cuckoo Filter.

B.4. Since draft-ietf-httpbis-cache-digest-01

- Added definition of the Cache-Digest header.
- Introduce ACCEPT_CACHE_DIGEST SETTINGS parameter.

- Change intended status from Standard to Experimental.

B.5. Since draft-ietf-httpbis-cache-digest-00

- Make the scope of a digest frame explicit and shift to stream 0.

Appendix C. Acknowledgements

+{:numbered="false"}

Thanks to Stefan Eissing for his suggestions.

Authors' Addresses

Kazuho Oku
Fastly

Email: kazuhooku@gmail.com

Yoav Weiss
Akamai

Email: yoav@yoav.ws
URI: https://blog.yoav.ws/

HTTP Working Group
Internet-Draft
Intended status: Experimental
Expires: January 4, 2021

I. Grigorik
Y. Weiss
Google

July 3, 2020

HTTP Client Hints
draft-ietf-httpbis-client-hints-15

Abstract

HTTP defines proactive content negotiation to allow servers to select the appropriate response for a given request, based upon the user agent's characteristics, as expressed in request headers. In practice, user agents are often unwilling to send those request headers, because it is not clear whether they will be used, and sending them impacts both performance and privacy.

This document defines an Accept-CH response header that servers can use to advertise their use of request headers for proactive content negotiation, along with a set of guidelines for the creation of such headers, colloquially known as "Client Hints."

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/client-hints [3].

Status of This Memo

This Internet-Draft will expire on January 4, 2021.

Table of Contents

1. Introduction 3
1.1. Notational Conventions 4
2. Client Hint Request Header Fields 4
2.1. Sending Client Hints 4
2.2. Server Processing of Client Hints 5
3. Advertising Server Support 5
3.1. The Accept-CH Response Header Field 5
3.2. Interaction with Caches 6
4. Security Considerations 7
4.1. Information Exposure 7
4.2. Deployment and Security Risks 9
4.3. Abuse Detection 9
5. Cost of Sending Hints 9
6. IANA Considerations 10
6.1. Accept-CH 10
7. References 10
7.1. Normative References 10
7.2. Informative References 11
7.3. URIs 11
Appendix A. Changes 11
A.1. Since -00 11
A.2. Since -01 12
A.3. Since -02 12
A.4. Since -03 12
A.5. Since -04 12
A.6. Since -05 12
A.7. Since -06 12
A.8. Since -07 12
A.9. Since -08 13

A.10. Since -09 13
A.11. Since -10 13
A.12. Since -11 13
A.13. Since -12 13
A.14. Since -13 13
A.15. Since -14 13
Acknowledgements 13
Authors' Addresses 13

1. Introduction

There are thousands of different devices accessing the web, each with different device capabilities and preference information. These device capabilities include hardware and software characteristics, as well as dynamic user and user agent preferences. Historically, applications that wanted the server to optimize content delivery and user experience based on such capabilities had to rely on passive identification (e.g., by matching the User-Agent header field (Section 5.5.3 of [RFC7231]) against an established database of user agent signatures), use HTTP cookies [RFC6265] and URL parameters, or use some combination of these and similar mechanisms to enable ad hoc content negotiation.

Such techniques are expensive to set up and maintain, and are not portable across both applications and servers. They also make it hard for both user agent and server to understand which data are required and is in use during the negotiation:

- User agent detection cannot reliably identify all static variables, cannot infer dynamic user agent preferences, requires an external device database, is not cache friendly, and is reliant on a passive fingerprinting surface.
- Cookie-based approaches are not portable across applications and servers, impose additional client-side latency by requiring JavaScript execution, and are not cache friendly.
- URL parameters, similar to cookie-based approaches, suffer from lack of portability, and are hard to deploy due to a requirement to encode content negotiation data inside of the URL of each resource.

Proactive content negotiation (Section 3.4.1 of [RFC7231]) offers an alternative approach; user agents use specified, well-defined request headers to advertise their capabilities and characteristics, so that servers can select (or formulate) an appropriate response based on those request headers (or on other, implicit characteristics).

However, traditional proactive content negotiation techniques often mean that user agents send these request headers prolifically. This

causes performance concerns (because it creates "bloat" in requests), as well as privacy issues; passively providing such information allows servers to silently fingerprint the user.

This document defines Client Hints, a framework that enables servers to opt-in to specific proactive content negotiation features, adapting their content accordingly, as well as guidelines for content negotiation mechanisms that use the framework. This document also defines a new response header, Accept-CH, that allows an origin server to explicitly ask that user agents send these headers in requests.

Client Hints mitigate performance concerns by assuring that user agents will only send the request headers when they're actually going to be used, and privacy concerns of passive fingerprinting by requiring explicit opt-in and disclosure of required headers by the server through the use of the Accept-CH response header, turning passive fingerprinting vectors into active ones.

The document does not define specific usages of Client Hints. Such usages need to be defined in their respective specifications.

One example of such usage is the User Agent Client Hints [UA-CH].

1.1. Notational Conventions

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.

This document uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].

2. Client Hint Request Header Fields

A Client Hint request header field is a HTTP header field that is used by HTTP user agents to indicate data that can be used by the server to select an appropriate response. Each one conveys user agent preferences that the server can use to adapt and optimize the response.

2.1. Sending Client Hints

User agents choose what Client Hints to send in a request based on their default settings, user configuration, and server preferences expressed in "Accept-CH". The user agent and server can use an opt-

in mechanism outlined below to negotiate which header fields need to be sent to allow for efficient content adaption, and optionally use additional mechanisms (e.g., as outlined in [CLIENT-HINTS-INFRASTRUCTURE]) to negotiate delegation policies that control access of third parties to those same header fields. User agents SHOULD require an opt-in to send any hints that are not listed in the low-entropy hint table at [CLIENT-HINTS-INFRASTRUCTURE].

Implementers need to be aware of the fingerprinting implications when implementing support for Client Hints, and follow the considerations outlined in the Security Considerations (Section 4) section of this document.

2.2. Server Processing of Client Hints

When presented with a request that contains one or more Client Hint header fields, servers can optimize the response based upon the information in them. When doing so, and if the resource is cacheable, the server MUST also generate a Vary response header field (Section 7.1.4 of [RFC7231]) to indicate which hints can affect the selected response and whether the selected response is appropriate for a later request.

Servers MUST ignore hints they do not understand nor support. There is no mechanism for servers to indicate to user agents that hints were ignored.

Furthermore, the server can generate additional response header fields (as specified by the hint or hints in use) that convey related values to aid client processing.

3. Advertising Server Support

Servers can advertise support for Client Hints using the mechanism described below.

3.1. The Accept-CH Response Header Field

The Accept-CH response header field indicates server support for the hints indicated in its value. Servers wishing to receive user agent information through Client Hints SHOULD add Accept-CH response header to their responses as early as possible.

Accept-CH is a Structured Header [I-D.ietf-httpbis-header-structure]. Its value MUST be an sf-list (Section 3.1 of [I-D.ietf-httpbis-header-structure]) whose members are tokens (Section 3.3.4 of [I-D.ietf-httpbis-header-structure]). Its ABNF is:

```
Accept-CH = sf-list
```

For example:

```
Accept-CH: Sec-CH-Example, Sec-CH-Example-2
```

When a user agent receives an HTTP response containing "Accept-CH", that indicates that the origin opts-in to receive the indicated request header fields for subsequent same-origin requests. The opt-in MUST be ignored if delivered over non-secure transport (using a scheme different from HTTPS). It SHOULD be persisted and bound to the origin to enable delivery of Client Hints on subsequent requests to the server's origin, for the duration of the user's session (as defined by the user agent). An opt-in overrides previous persisted opt-in values and SHOULD be persisted in its stead.

Based on the Accept-CH example above, which is received in response to a user agent navigating to "https://site.example", and delivered over a secure transport, persisted Accept-CH preferences will be bound to "https://site.example". It will then use it for navigations to e.g., "https://site.example/foobar.html", but not to e.g., "https://foobar.site.example/". It will similarly use the preference for any same-origin resource requests (e.g., to "https://site.example/image.jpg") initiated by the page constructed from the navigation's response, but not to cross-origin resource requests (e.g., "https://thirdparty.example/resource.js"). This preference will not extend to resource requests initiated to "https://site.example" from other origins (e.g., from navigations to "https://other.example/").

3.2. Interaction with Caches

When selecting a response based on one or more Client Hints, and if the resource is cacheable, the server needs to generate a Vary response header field ([RFC7234]) to indicate which hints can affect the selected response and whether the selected response is appropriate for a later request.

```
Vary: Sec-CH-Example
```

The above example indicates that the cache key needs to include the Sec-CH-Example header field.

```
Vary: Sec-CH-Example, Sec-CH-Example-2
```

The above example indicates that the cache key needs to include the Sec-CH-Example and Sec-CH-Example-2 header fields.

4. Security Considerations

4.1. Information Exposure

Request header fields used in features relying on this document expose information about the user's environment to enable privacy-preserving proactive content negotiation, and avoid exposing passive fingerprinting vectors. However, implementers need to bear in mind that in the worst case, uncontrolled and unmonitored active fingerprinting is not better than passive fingerprinting. In order to provide user privacy benefits, user agents need to apply further policies that prevent abuse of the information exposed by features using Client Hints.

The information exposed by features might reveal new information about the user and implementers ought to consider the following considerations, recommendations, and best practices.

The underlying assumption is that exposing information about the user as a request header is equivalent (from a security perspective) to exposing this information by other means. (For example, if the request's origin can access that information using JavaScript APIs, and transmit it to its servers).

Because Client Hints is an explicit opt-in mechanism, that means that servers that want access to information about the user's environment need to actively ask for it, enabling clients and privacy researchers to keep track of which origins collect that data, and potentially act upon it. The header-based opt-in means that removal of passive fingerprinting vectors is possible, such as the User-Agent string (enabling active access to that information through User-Agent Client Hints ([UA-CH]) or otherwise expose information already available through script (e.g., the Save-Data Client Hint [4]), without increasing the passive fingerprinting surface. User agents supporting Client Hints features which send certain information to opted-in servers SHOULD avoid sending the equivalent information passively.

Therefore, features relying on this document to define Client Hint headers MUST NOT provide new information that is otherwise not made available to the application by the user agent, such as existing request headers, HTML, CSS, or JavaScript.

Such features need to take into account the following aspects of the information exposed:

- Entropy - Exposing highly granular data can be used to help identify users across multiple requests to different origins.

Reducing the set of header field values that can be expressed, or restricting them to an enumerated range where the advertised value is close to but is not an exact representation of the current value, can improve privacy and reduce risk of linkability by ensuring that the same value is sent by multiple users.

- Sensitivity - The feature SHOULD NOT expose user-sensitive information. To that end, information available to the application, but gated behind specific user actions (e.g., a permission prompt or user activation) SHOULD NOT be exposed as a Client Hint.
- Change over time - The feature SHOULD NOT expose user information that changes over time, unless the state change itself is also exposed (e.g., through JavaScript callbacks).

Different features will be positioned in different points in the space between low-entropy, non-sensitive and static information (e.g., user agent information), and high-entropy, sensitive and dynamic information (e.g., geolocation). User agents need to consider the value provided by a particular feature vs these considerations, and may wish to have different policies regarding that tradeoff on a per-feature or other fine-grained basis.

Implementers ought to consider both user- and server- controlled mechanisms and policies to control which Client Hints header fields are advertised:

- Implementers SHOULD restrict delivery of some or all Client Hints header fields to the opt-in origin only, unless the opt-in origin has explicitly delegated permission to another origin to request Client Hints header fields.
- Implementers considering providing user choice mechanisms that allow users to balance privacy concerns against bandwidth limitations need to also consider that explaining to users the privacy implications involved, such as the risks of passive fingerprinting, may be challenging or even impractical.
- Implementations specific to certain use cases or threat models MAY avoid transmitting some or all of Client Hints header fields. For example, avoid transmission of header fields that can carry higher risks of linkability.

User agents MUST clear persisted opt-in preferences when any one of site data, browsing history, browsing cache, cookies, or similar, are cleared.

4.2. Deployment and Security Risks

Deployment of new request headers requires several considerations:

- Potential conflicts due to existing use of header field name
- Properties of the data communicated in header field value

Authors of new Client Hints are advised to carefully consider whether they need to be able to be added by client-side content (e.g., scripts), or whether they need to be exclusively set by the user agent. In the latter case, the Sec- prefix on the header field name has the effect of preventing scripts and other application content from setting them in user agents. Using the "Sec-" prefix signals to servers that the user agent - and not application content - generated the values. See [FETCH] for more information.

By convention, request headers that are Client Hints are encouraged to use a CH- prefix, to make them easier to identify as using this framework; for example, CH-Foo or, with a "Sec-" prefix, Sec-CH-Foo. Doing so makes them easier to identify programmatically (e.g., for stripping unrecognised hints from requests by privacy filters).

A Client Hints request header negotiated using the Accept-CH opt-in mechanism MUST have a field name that matches sf-token (Section 3.3.4 of [I-D.ietf-httpbis-header-structure]).

4.3. Abuse Detection

A user agent that tracks access to active fingerprinting information SHOULD consider emission of Client Hints headers similarly to the way it would consider access to the equivalent API.

Research into abuse of Client Hints might look at how HTTP responses to requests that contain Client Hints differ from those with different values, and from those without. This might be used to reveal which Client Hints are in use, allowing researchers to further analyze that use.

5. Cost of Sending Hints

Sending Client Hints to the server incurs an increase in request byte size. Some of this increase can be mitigated by HTTP header compression schemes, but each new hint sent will still lead to some increased bandwidth usage. Servers SHOULD take that into account when opting in to receive Client Hints, and SHOULD NOT opt-in to receive hints unless they are to be used for content adaptation purposes.

Due to request byte size increase, features relying on this document to define Client Hints MAY consider restricting sending those hints to certain request destinations [FETCH], where they are more likely to be useful.

6. IANA Considerations

Features relying on this document are expected to register added request header fields in the Permanent Message Header Fields registry ([RFC3864]).

This document defines the "Accept-CH" HTTP response header field, and registers it in the same registry.

6.1. Accept-CH

- Header field name: Accept-CH
- Applicable protocol: HTTP
- Status: experimental
- Author/Change controller: IETF
- Specification document(s): Section 3.1 of this document
- Related information: for Client Hints

7. References

7.1. Normative References

[CLIENT-HINTS-INFRASTRUCTURE]
Weiss, Y., "Client Hints Infrastructure", n.d., <https://wicg.github.io/client-hints-infrastructure/>.

[I-D.ietf-httpbis-header-structure]
Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", draft-ietf-httpbis-header-structure-19 (work in progress), June 2020.

[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>.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/info/rfc3864>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <https://www.rfc-editor.org/info/rfc7234>.

[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

[FETCH] van Kesteren, A., "Fetch", n.d., <https://fetch.spec.whatwg.org/>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[UA-CH] West, M. and Y. Weiss, "User Agent Client Hints", n.d., <https://wicg.github.io/ua-client-hints/>.

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/client-hints

[4] https://wicg.github.io/savedata/#save-data-request-header-field

Appendix A. Changes

A.1. Since -00

- Issue 168 (make Save-Data extensible) updated ABNF.
- Issue 163 (CH review feedback) editorial feedback from httpwg list.

- Issue 153 (NetInfo API citation) added normative reference.

A.2. Since -01

- Issue 200: Moved Key reference to informative.
- Issue 215: Extended passive fingerprinting and mitigation considerations.
- Changed document status to experimental.

A.3. Since -02

- Issue 239: Updated reference to CR-css-values-3
- Issue 240: Updated reference for Network Information API
- Issue 241: Consistency in IANA considerations
- Issue 250: Clarified Accept-CH

A.4. Since -03

- Issue 284: Extended guidance for Accept-CH
- Issue 308: Editorial cleanup
- Issue 306: Define Accept-CH-Lifetime

A.5. Since -04

- Issue 361: Removed Downlink
- Issue 361: Moved Key to appendix, plus other editorial feedback

A.6. Since -05

- Issue 372: Scoped CH opt-in and delivery to secure transports
- Issue 373: Bind CH opt-in to origin

A.7. Since -06

- Issue 524: Save-Data is now defined by NetInfo spec, dropping
- PR 775: Removed specific features to be defined in other specifications

A.8. Since -07

- Issue 761: Clarified that the defined headers are response headers.
- Issue 730: Replaced Key reference with Variants.
- Issue 700: Replaced ABNF with structured headers.
- PR 878: Removed Accept-CH-Lifetime based on feedback at IETF 105

A.9. Since -08

- PR 985: Describe the bytesize cost of hints.
- PR 776: Add Sec- and CH- prefix considerations.
- PR 1001: Clear CH persistence when cookies are cleared.

A.10. Since -09

- PR 1064: Fix merge issues with "cost of sending hints".

A.11. Since -10

- PR 1072: LC feedback from Julian Reschke.
- PR 1080: Improve list style.
- PR 1082: Remove section mentioning Variants.
- PR 1097: Editorial feedback from mnot.
- PR 1131: Remove unused references.
- PR 1132: Remove nested list.

A.12. Since -11

- PR 1134: Re-insert back section.

A.13. Since -12

- PR 1160: AD review.

A.14. Since -13

- PR 1171: Genart review.

A.15. Since -14

- PR 1220: AD review.

Acknowledgements

Thanks to Mark Nottingham, Julian Reschke, Chris Bentzel, Ben Greenstein, Tarun Bansal, Roy Fielding, Vasiliy Faronov, Ted Hardie, Jonas Sicking, Martin Thomson, and numerous other members of the IETF HTTP Working Group for invaluable help and feedback.

Authors' Addresses

Ilya Grigorik
Google

Email: ilya@igvita.com
URI: https://www.igvita.com/

Yoav Weiss
Google

Email: yoav@yoav.ws
URI: https://blog.yoav.ws/

HTTP Working Group
Internet-Draft
Updates: 6265 (if approved)
Intended status: Standards Track
Expires: March 9, 2017

M. West
Google, Inc

September 5, 2016

Deprecate modification of ’secure’ cookies from non-secure origins
draft-ietf-httpbis-cookie-alone-01

Abstract

This document updates RFC6265 by removing the ability for a non-secure origin to set cookies with a ’secure’ flag, and to overwrite cookies whose ’secure’ flag is set. This deprecation improves the isolation between HTTP and HTTPS origins, and reduces the risk of malicious interference.

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/ .

Working Group information can be found at http://httpwg.github.io/ ; source code and issues list for this draft can be found at https://github.com/httpwg/http-extensions/labels/cookie-alone .

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 March 9, 2017.

Table of Contents

1. Introduction 2
2. Terminology and notation 3
3. Recommendations 3
4. Security Considerations 4
5. References 4
5.1. Normative References 4
5.2. Informative References 5
Appendix A. Acknowledgements 5
Appendix B. Changes 5
B.1. Since -00 6
Author's Address 6

1. Introduction

Section 8.5 and Section 8.6 of [RFC6265] spell out some of the drawbacks of cookies' implementation: due to historical accident, non-secure origins can set cookies which will be delivered to secure origins in a manner indistinguishable from cookies set by that origin itself. This enables a number of attacks, which have been recently spelled out in some detail in [COOKIE-INTEGRITY].

We can mitigate the risk of these attacks by making it more difficult for non-secure origins to influence the state of secure origins. Accordingly, this document recommends the deprecation and removal of non-secure origins' ability to write cookies with a 'secure' flag, and their ability to overwrite cookies whose 'secure' flag is set.

2. Terminology and notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The "scheme" component of a URI is defined in Section 3 of [RFC3986].

3. Recommendations

This document updates Section 5.3 of [RFC6265] as follows:

1. After step 8 of the current algorithm, which sets the cookie's "secure-only-flag", execute the following step:

    1. If the "scheme" component of the "request-uri" does not denote a "secure" protocol (as defined by the user agent), and the cookie's "secure-only-flag" is "true", then abort these steps and ignore the newly created cookie entirely.
2. Before step 11, execute the following step:

    1. If the newly created cookie's "secure-only-flag" is not set, and the "scheme" component of the "request-uri" does not denote a "secure" protocol, then abort these steps and ignore the newly created cookie entirely if the cookie store contains one or more cookies that meet all of the following criteria:

        1. Their "name" matches the "name" of the newly created cookie.
        2. Their "secure-only-flag" is set.
        3. Their "domain" domain-matches the "domain" of the newly created cookie, or vice-versa.
        4. The "path" of the newly created cookie path-matches the "path" of the existing cookie.

        Note: The "path" comparison is not symmetric, ensuring only that a newly-created non-secure cookie does not overlay an existing secure cookie, providing some mitigation against cookie fixing attacks. That is, given an existing secure cookie named "a" with a "path" of "/login", a non-secure cookie named "a" could be set for a "path" of "/" or "/foo", but not for a "path" of "/login" or "/login/en".

        Note: This allows "secure" pages to override "secure" cookies with non-secure variants. Perhaps we should restrict that as well?

3. In order to ensure that a non-secure site can never cause a "secure" cookie to be evicted, adjust the "remove excess cookies" priority order at the bottom of Section 5.3 to be the following:

    1. Expired cookies.
    2. Cookies whose "secure-only-flag" is not set and which share a "domain" field with more than a predetermined number of other cookies.
    3. Cookies that share a "domain" field with more than a predetermined number of other cookies.
    4. All cookies.

    Note that the eviction algorithm specified here is triggered only after insertion of a cookie which causes the user agent to exceed some predetermined upper bound. Conforming user agents MUST ensure that inserting a non-secure cookie does not cause a secure cookie to be removed.

4. Security Considerations

This specification increases a site's confidence that secure cookies it sets will remain unmodified by insecure pages on hosts which it domain-matches. Ideally, sites would use HSTS as described in [RFC6797] to defend more robustly against the dangers of non-secure transport in general, but until adoption of that protection becomes ubiquitous, this deprecation this document recommends will mitigate a number of risks.

The mitigations in this document do not, however, give complete confidence that a given cookie was set securely. If an attacker is able to impersonate a response from "http://example.com/" before a user visits "https://example.com/", the user agent will accept any cookie that the insecure origin sets, as the "secure" cookie won't yet be present in the user agent's cookie store. An active network attacker may still be able to use this ability to mount an attack against "example.com", even if that site uses HTTPS exclusively.

The proposal in [COOKIE-PREFIXES] could mitigate this risk, as could "preloading" HSTS for "example.com" into the user agent [HSTS-PRELOADING].

5. References

5.1. Normative References

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <http://www.rfc-editor.org/info/rfc6265>.

5.2. Informative References

[COOKIE-INTEGRITY] Zheng, X., Jiang, J., Liang, J., Duan, H., Chen, S., Wan, T., and N. Weaver, "Cookies Lack Integrity: Real-World Implications", August 2015, <https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/zheng>.

[COOKIE-PREFIXES] West, M., "Cookie Prefixes", 2016, <https://tools.ietf.org/html/draft-ietf-httpbis-cookie-prefixes>.

[HSTS-PRELOADING] "HSTS Preload Submission", n.d., <https://hstspreload.appspot.com/>.

[RFC6797] Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)", RFC 6797, DOI 10.17487/RFC6797, November 2012, <http://www.rfc-editor.org/info/rfc6797>.

Appendix A. Acknowledgements

Richard Barnes encouraged a formalization of the deprecation proposal. [COOKIE-INTEGRITY] was a useful exploration of the issues [RFC6265] described.

Appendix B. Changes

B.1. Since -00

- Issue 223 addressed by adding a path-match constraint to the storage algorithm for non-secure cookies. This ensures that non-secure cookies cannot overlay secure cookies for a given path, but allows secure and non-secure cookies with the same name to exist on distinct paths.

Author's Address

Mike West
Google, Inc

Email: mkwst@google.com
URI: https://mikewest.org/

HTTP Working Group
Internet-Draft
Updates: 6265 (if approved)
Intended status: Standards Track
Expires: August 26, 2016

M. West
Google, Inc

February 23, 2016

Cookie Prefixes
draft-ietf-httpbis-cookie-prefixes-00

Abstract

This document updates RFC6265 by adding a set of restrictions upon the names which may be used for cookies with specific properties. These restrictions enable user agents to smuggle cookie state to the server within the confines of the existing "Cookie" request header syntax, and limits the ways in which cookies may be abused in a conforming user agent.

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 August 26, 2016.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology and notation 2
3. Prefixes 3
3.1. The "__Secure-" prefix 3
3.2. The "__Host-" prefix 3
4. User Agent Requirements 4
5. Aesthetic Considerations 4
5.1. Not pretty. 4
5.2. Why "__"? 4
6. Security Considerations 4
6.1. Secure Origins Only 5
6.2. Limitations 5
7. References 5
7.1. Normative References 5
7.2. Informative References 5
Appendix A. Acknowledgements 6
Author's Address 6

1. Introduction

Section 8.5 and Section 8.6 of [RFC6265] spell out some of the drawbacks of cookies' implementation: due to historical accident, it is impossible for a server to have confidence that a cookie set in a secure way (e.g., as a domain cookie with the "Secure" (and possibly "HttpOnly") flags set) remains intact and untouched by non-secure subdomains.

We can't alter the syntax of the "Cookie" request header, as that would likely break a number of implementations. This rules out sending a cookie's flags along with the cookie directly, but we can smuggle information along with the cookie if we reserve certain name prefixes for cookies with certain properties.

This document describes such a scheme, which enables servers to set cookies which conforming user agents will ensure are "Secure", and locked to a domain.

2. Terminology and notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The "scheme" component of a URI is defined in Section 3 of [RFC3986].

3. Prefixes

3.1. The "__Secure-" prefix

If a cookie's name begins with "__Secure-", the cookie MUST be:

1. Set with a "Secure" attribute
2. Set from a URI whose "scheme" is considered "secure" by the user agent.

The following cookie would be rejected when set from any origin, as the "Secure" flag is not set

```
Set-Cookie: __Secure-SID=12345; Domain=example.com
```

While the following would be accepted if set from a secure origin (e.g. "https://example.com/"), and rejected otherwise:

```
Set-Cookie: __Secure-SID=12345; Secure; Domain=example.com
```

3.2. The "__Host-" prefix

If a cookie's name begins with "__Host-", the cookie MUST be:

1. Set with a "Secure" attribute
2. Set from a URI whose "scheme" is considered "secure" by the user agent.
3. Sent only to the host which set the cookie. That is, a cookie named "__Host-cookie1" set from "https://example.com" MUST NOT contain a "Domain" attribute (and will therefore be sent only to "example.com", and not to "subdomain.example.com").
4. Sent to every request for a host. That is, a cookie named "__Host-cookie1" MUST contain a "Path" attribute with a value of "/".

The following cookies would always be rejected:

```
Set-Cookie: __Host-SID=12345
Set-Cookie: __Host-SID=12345; Secure
Set-Cookie: __Host-SID=12345; Domain=example.com
Set-Cookie: __Host-SID=12345; Domain=example.com; Path=/
Set-Cookie: __Host-SID=12345; Secure; Domain=example.com; Path=/
```

While the following would be accepted if set from a secure origin (e.g. "https://example.com/"), and rejected otherwise:

```
Set-Cookie: __Host-SID=12345; Secure; Path=/
```

4. User Agent Requirements

This document updates Section 5.3 of [RFC6265] as follows:

After step 10 of the current algorithm, the cookies flags are set. Insert the following steps to perform the prefix checks this document specifies:

1. If the "cookie-name" begins with the string "__Secure-" or "__Host-", abort these steps and ignore the cookie entirely unless both of the following conditions are true:
   * The cookie's "secure-only-flag" is "true"
   * "request-uri"'s "scheme" component denotes a "secure" protocol (as determined by the user agent)
2. If the "cookie-name" begins with the string "__Host-", abort these steps and ignore the cookie entirely unless the following conditions are true:
   * The cookie's "host-only-flag" is "true"
   * The cookie's "path" is "/"

5. Aesthetic Considerations

5.1. Not pretty.

Prefixes are ugly. :(

5.2. Why "__"?

We started with "$", but ran into issues with servers that had implemented [RFC2109]-style cookies. "__" is a prefix used for a number of well-known cookies in the wild (notably Google Analytics's "__ut*" cookies, and CloudFlare's "__cfduid"), and so is unlikely to produce such compatibility issues, while being uncommon enough to mitigate the risk of collisions.

6. Security Considerations

6.1. Secure Origins Only

It would certainly be possible to extend this scheme to non-secure origins (and an earlier draft of this document did exactly that). User agents, however, are slowly moving towards a world where features with security implications are available only over secure transport (see [SECURE-CONTEXTS], [POWERFUL-FEATURES], and [DEPRECATING-HTTP]). This document follows that trend, limiting exciting new cookie properties to secure transport in order to ensure that user agents can make claims which middlemen will have a hard time violating.

To that end, note that the requirements listed above mean that prefixed cookies will be rejected entirely if a non-secure origin attempts to set them.

6.2. Limitations

This scheme gives no assurance to the server that the restrictions on cookie names are enforced. Servers could certainly probe the user agent's functionality to determine support, or sniff based on the "User-Agent" request header, if such assurances were deemed necessary.

7. References

7.1. Normative References

[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>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <http://www.rfc-editor.org/info/rfc6265>.

7.2. Informative References

[DEPRECATING-HTTP] Barnes, R., "Deprecating Non-Secure HTTP", April 2015, <https://blog.mozilla.org/security/2015/04/30/deprecating-non-secure-http/>.

[Lawrence2015]
Lawrence, E., "Duct Tape and Baling Wire -- Cookie Prefixes", October 2015, <http://textslashplain.com/2015/10/09/duct-tape-and-baling-wirecookie-prefixes/>.

[POWERFUL-FEATURES]
Palmer, C., "Prefer Secure Origins for Powerful New Features", 2015, <https://www.chromium.org/Home/chromium-security/prefer-secure-origins-for-powerful-new-features>.

[RFC2109] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2109, DOI 10.17487/RFC2109, February 1997, <http://www.rfc-editor.org/info/rfc2109>.

[SECURE-CONTEXTS]
West, M., "Secure Contexts", 2016, <https://w3c.github.io/webappsec-secure-contexts/>.

Appendix A. Acknowledgements

Eric Lawrence had this idea a million years ago, and wrote about its genesis in [Lawrence2015]. Devdatta Akhawe helped justify the potential impact of the scheme on real-world websites. Thomas Broyer pointed out the issues with a leading "$" in the prefixes, and Brian Smith provided valuable contributions to the discussion around a replacement (ISO C indeed).

Author’s Address

Mike West
Google, Inc

Email: mkwst@google.com
URI: https://mikewest.org/

HTTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 20, 2017

M. Thomson
Mozilla

April 18, 2017

Encrypted Content-Encoding for HTTP
draft-ietf-httpbis-encryption-encoding-09

Abstract

This memo introduces a content coding for HTTP that allows message payloads to be encrypted.

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/ .

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

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 October 20, 2017.

Table of Contents

1. Introduction 2
1.1. Notational Conventions 3
2. The "aes128gcm" HTTP Content Coding 3
2.1. Encryption Content Coding Header 5
2.2. Content Encryption Key Derivation 6
2.3. Nonce Derivation 6
3. Examples 7
3.1. Encryption of a Response 7
3.2. Encryption with Multiple Records 8
4. Security Considerations 8
4.1. Automatic Decryption 9
4.2. Message Truncation 9
4.3. Key and Nonce Reuse 9
4.4. Data Encryption Limits 9
4.5. Content Integrity 10
4.6. Leaking Information in Header Fields 10
4.7. Poisoning Storage 11
4.8. Sizing and Timing Attacks 11
5. IANA Considerations 12
5.1. The "aes128gcm" HTTP Content Coding 12
6. References 12
6.1. Normative References 12
6.2. Informative References 13
Appendix A. JWE Mapping 14
Appendix B. Acknowledgements 15
Author’s Address 15

1. Introduction

It is sometimes desirable to encrypt the contents of a HTTP message (request or response) so that when the payload is stored (e.g., with a HTTP PUT), only someone with the appropriate key can read it.

For example, it might be necessary to store a file on a server without exposing its contents to that server. Furthermore, that same file could be replicated to other servers (to make it more resistant to server or network failure), downloaded by clients (to make it available offline), etc. without exposing its contents.

These uses are not met by the use of TLS [RFC5246], since it only encrypts the channel between the client and server.

This document specifies a content coding (Section 3.1.2 of [RFC7231]) for HTTP to serve these and other use cases.

This content coding is not a direct adaptation of message-based encryption formats - such as those that are described by [RFC4880], [RFC5652], [RFC7516], and [XMLENC] - which are not suited to stream processing, which is necessary for HTTP. The format described here follows more closely to the lower level constructs described in [RFC5116].

To the extent that message-based encryption formats use the same primitives, the format can be considered as sequence of encrypted messages with a particular profile. For instance, Appendix A explains how the format is congruent with a sequence of JSON Web Encryption [RFC7516] values with a fixed header.

This mechanism is likely only a small part of a larger design that uses content encryption. How clients and servers acquire and identify keys will depend on the use case. In particular, a key management system is not described.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. The "aes128gcm" HTTP Content Coding

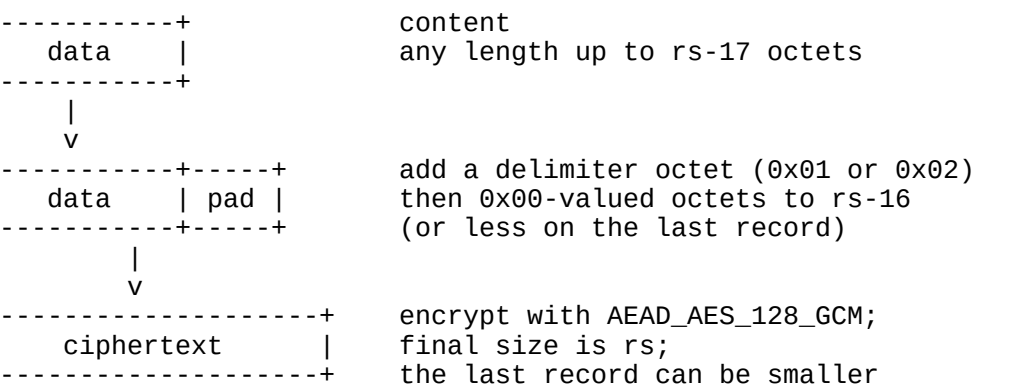
The "aes128gcm" HTTP content coding indicates that a payload has been encrypted using Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) as identified as AEAD_AES_128_GCM in [RFC5116], Section 5.1. The AEAD_AES_128_GCM algorithm uses a 128 bit content encryption key.

Using this content coding requires knowledge of a key. How this key is acquired is not defined in this document.

The "aes128gcm" content coding uses a single fixed set of encryption primitives. Cipher suite agility is achieved by defining a new content coding scheme. This ensures that only the HTTP Accept-Encoding header field is necessary to negotiate the use of encryption.

The "aes128gcm" content coding uses a fixed record size. The final encoding consists of a header (see Section 2.1) and zero or more fixed size encrypted records; the final record can be smaller than the record size.

The record size determines the length of each portion of plaintext that is enciphered. The record size ("rs") is included in the content coding header (see Section 2.1).



AEAD_AES_128_GCM produces ciphertext 16 octets longer than its input plaintext. Therefore, the unencrypted content of each record is shorter than the record size by 16 octets. Valid records always contain at least a padding delimiter octet and a 16 octet authentication tag.

Each record contains a single padding delimiter octet followed by any number of zero octets. The last record uses a padding delimiter octet set to the value 2, all other records have a padding delimiter octet value of 1.

On decryption, the padding delimiter is the last non-zero valued octet of the record. A decrypter MUST fail if the record contains no non-zero octet. A decrypter MUST fail if the last record contains a padding delimiter with a value other than 2 or if any record other than the last contains a padding delimiter with a value other than 1.

The nonce for each record is a 96-bit value constructed from the record sequence number and the input keying material. Nonce derivation is covered in Section 2.3.

The additional data passed to each invocation of AEAD_AES_128_GCM is a zero-length octet sequence.

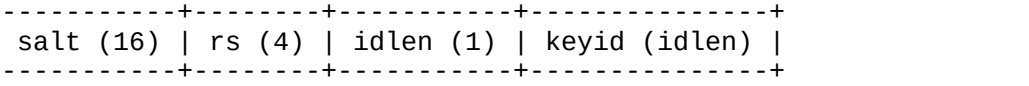
A consequence of this record structure is that range requests [RFC7233] and random access to encrypted payload bodies are possible at the granularity of the record size. Partial records at the ends of a range cannot be decrypted. Thus, it is best if range requests start and end on record boundaries. Note however that random access to specific parts of encrypted data could be confounded by the presence of padding.

Selecting the record size most appropriate for a given situation requires a trade-off. A smaller record size allows decrypted octets to be released more rapidly, which can be appropriate for applications that depend on responsiveness. Smaller records also reduce the additional data required if random access into the ciphertext is needed.

Applications that don't depending on streaming, random access, or arbitrary padding can use larger records, or even a single record. A larger record size reduces processing and data overheads.

2.1. Encryption Content Coding Header

The content coding uses a header block that includes all parameters needed to decrypt the content (other than the key). The header block is placed in the body of a message ahead of the sequence of records.



salt: The "salt" parameter comprises the first 16 octets of the "aes128gcm" content coding header. The same "salt" parameter value MUST NOT be reused for two different payload bodies that have the same input keying material; generating a random salt for every application of the content coding ensures that content encryption key reuse is highly unlikely.

rs: The "rs" or record size parameter contains an unsigned 32-bit integer in network byte order that describes the record size in octets. Note that it is therefore impossible to exceed the 2^36-31 limit on plaintext input to AEAD_AES_128_GCM. Values smaller than 18 are invalid.

idlen: The "idlen" parameter is an unsigned 8-bit integer that defines the length of the "keyid" parameter.

keyid: The "keyid" parameter can be used to identify the keying material that is used. This field is the length determined by the "idlen" parameter. Recipients that receive a message are expected

to know how to retrieve keys; the "keyid" parameter might be input to that process. A "keyid" parameter SHOULD be a UTF-8 [RFC3629] encoded string, particularly where the identifier might need to be rendered in a textual form.

2.2. Content Encryption Key Derivation

In order to allow the reuse of keying material for multiple different HTTP messages, a content encryption key is derived for each message. The content encryption key is derived from the "salt" parameter using the HMAC-based key derivation function (HKDF) described in [RFC5869] using the SHA-256 hash algorithm [FIPS180-4].

The value of the "salt" parameter is the salt input to HKDF function. The keying material identified by the "keyid" parameter is the input keying material (IKM) to HKDF. Input keying material is expected to be provided to recipients separately. The extract phase of HKDF therefore produces a pseudorandom key (PRK) as follows:

```
PRK = HMAC-SHA-256(salt, IKM)
```

The info parameter to HKDF is set to the ASCII-encoded string "Content-Encoding: aes128gcm" and a single zero octet:

```
cek_info = "Content-Encoding: aes128gcm" || 0x00
```

Note(1): Concatenation of octet sequences is represented by the "||" operator.

Note(2): The strings used here and in Section 2.3 do not include a terminating 0x00 octet, as is used in some programming languages.

AEAD_AES_128_GCM requires a 16 octet (128 bit) content encryption key (CEK), so the length (L) parameter to HKDF is 16. The second step of HKDF can therefore be simplified to the first 16 octets of a single HMAC:

```
CEK = HMAC-SHA-256(PRK, cek_info || 0x01)
```

2.3. Nonce Derivation

The nonce input to AEAD_AES_128_GCM is constructed for each record. The nonce for each record is a 12 octet (96 bit) value that is derived from the record sequence number, input keying material, and salt.

The input keying material and salt values are input to HKDF with different info and length parameters.

The length (L) parameter is 12 octets. The info parameter for the nonce is the ASCII-encoded string "Content-Encoding: nonce", terminated by a a single zero octet:

```
nonce_info = "Content-Encoding: nonce" || 0x00
```

The result is combined with the record sequence number - using exclusive or - to produce the nonce. The record sequence number (SEQ) is a 96-bit unsigned integer in network byte order that starts at zero.

Thus, the final nonce for each record is a 12 octet value:

```
NONCE = HMAC-SHA-256(PRK, nonce_info || 0x01) XOR SEQ
```

This nonce construction prevents removal or reordering of records.

3. Examples

This section shows a few examples of the encrypted content coding.

Note: All binary values in the examples in this section use Base 64 Encoding with URL and Filename Safe Alphabet [RFC4648]. This includes the bodies of requests. Whitespace and line wrapping is added to fit formatting constraints.

3.1. Encryption of a Response

Here, a successful HTTP GET response has been encrypted. This uses a record size of 4096 and no padding (just the single octet padding delimiter), so only a partial record is present. The input keying material is identified by an empty string (that is, the "keyid" field in the header is zero octets in length).

The encrypted data in this example is the UTF-8 encoded string "I am the walrus". The input keying material is the value "yqdlZ-tYemfogSmv7Ws5PQ" (in base64url). The 54 octet content body contains a single record and is shown here using 71 base64url characters for presentation reasons.

```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Length: 54
Content-Encoding: aes128gcm

I1BsxtFttlv3u_Oo94xnmwAAEAAA-NAVub2qFgBEuQKRapoZu-IxkIva3MEB1PD-
ly8Thjg
```

Note that the media type has been changed to "application/octet-stream" to avoid exposing information about the content. Alternatively (and equivalently), the Content-Type header field can be omitted.

Intermediate values for this example (all shown using base64url):

```
salt (from header) = I1BsxtFttlv3u_Oo94xnmw
PRK = zyeH5phsIsgUyd4oiSEIy35x-gIi4aM7y0hCF8mwn9g
CEK = _wniytB-ofscZDh4tbSjHw
NONCE = Bcs8gkIRKLI8GeI8
unencrypted data = SSBhbSB0aGUgd2FscnVzAg
```

3.2. Encryption with Multiple Records

This example shows the same message with input keying material of "BO3ZVPxUlnLORbVGMpbT1Q". In this example, the plaintext is split into records of 25 octets each (that is, the "rs" field in the header is 25). The first record includes one 0x00 padding octet. This means that there are 7 octets of message in the first record, and 8 in the second. A key identifier of the UTF-8 encoded string "a1" is also included in the header.

```
HTTP/1.1 200 OK
Content-Length: 73
Content-Encoding: aes128gcm

uNCkWiNYzKTnBN9ji3-qWAAAABkCYTHOG8chz_gnvgOqdGYovxyjuqRyJFjEDyoF
1Fvkj6hQPdPHI51OEUKEpgz3SsLWIqS_uA
```

4. Security Considerations

This mechanism assumes the presence of a key management framework that is used to manage the distribution of keys between valid senders and receivers. Defining key management is part of composing this mechanism into a larger application, protocol, or framework.

Implementation of cryptography - and key management in particular - can be difficult. For instance, implementations need to account for the potential for exposing keying material on side channels, such as might be exposed by the time it takes to perform a given operation. The requirements for a good implementation of cryptographic algorithms can change over time.

4.1. Automatic Decryption

As a content coding, a "aes128gcm" content coding might be automatically removed by a receiver in way that is not obvious to the ultimate consumer of a message. Recipients that depend on content origin authentication using this mechanism MUST reject messages that don't include the "aes128gcm" content coding.

4.2. Message Truncation

This content encoding is designed to permit the incremental processing of large messages. It also permits random access to plaintext in a limited fashion. The content encoding permits a receiver to detect when a message is truncated.

A partially delivered message MUST NOT be processed as though the entire message was successfully delivered. For instance, a partially delivered message cannot be cached as though it were complete.

An attacker might exploit willingness to process partial messages to cause a receiver to remain in a specific intermediate state. Implementations performing processing on partial messages need to ensure that any intermediate processing states don't advantage an attacker.

4.3. Key and Nonce Reuse

Encrypting different plaintext with the same content encryption key and nonce in AES-GCM is not safe [RFC5116]. The scheme defined here uses a fixed progression of nonce values. Thus, a new content encryption key is needed for every application of the content coding. Since input keying material can be reused, a unique "salt" parameter is needed to ensure a content encryption key is not reused.

If a content encryption key is reused - that is, if input keying material and salt are reused - this could expose the plaintext and the authentication key, nullifying the protection offered by encryption. Thus, if the same input keying material is reused, then the salt parameter MUST be unique each time. This ensures that the content encryption key is not reused. An implementation SHOULD generate a random salt parameter for every message.

4.4. Data Encryption Limits

There are limits to the data that AEAD_AES_128_GCM can encipher. The maximum value for the record size is limited by the size of the "rs" field in the header (see Section 2.1), which ensures that the 2^36-31 limit for a single application of AEAD_AES_128_GCM is not reached

[RFC5116]. In order to preserve a 2^-40 probability of indistinguishability under chosen plaintext attack (IND-CPA), the total amount of plaintext that can be enciphered with the key derived from the same input keying material and salt MUST be less than 2^44.5 blocks of 16 octets [AEBounds].

If the record size is a multiple of 16 octets, this means 398 terabytes can be encrypted safely, including padding and overhead. However, if the record size is not a multiple of 16 octets, the total amount of data that can be safely encrypted is reduced because partial AES blocks are encrypted. The worst case is a record size of 18 octets, for which at most 74 terabytes of plaintext can be encrypted, of which at least half is padding.

4.5. Content Integrity

This mechanism only provides content origin authentication. The authentication tag only ensures that an entity with access to the content encryption key produced the encrypted data.

Any entity with the content encryption key can therefore produce content that will be accepted as valid. This includes all recipients of the same HTTP message.

Furthermore, any entity that is able to modify both the Content-Encoding header field and the HTTP message body can replace the contents. Without the content encryption key or the input keying material, modifications to or replacement of parts of a payload body are not possible.

4.6. Leaking Information in Header Fields

Because only the payload body is encrypted, information exposed in header fields is visible to anyone who can read the HTTP message. This could expose side-channel information.

For example, the Content-Type header field can leak information about the payload body.

There are a number of strategies available to mitigate this threat, depending upon the application’s threat model and the users’ tolerance for leaked information:

1. Determine that it is not an issue. For example, if it is expected that all content stored will be "application/json", or another very common media type, exposing the Content-Type header field could be an acceptable risk.

2. If it is considered sensitive information and it is possible to determine it through other means (e.g., out of band, using hints in other representations, etc.), omit the relevant headers, and/or normalize them. In the case of Content-Type, this could be accomplished by always sending Content-Type: application/octet-stream (the most generic media type), or no Content-Type at all.

3. If it is considered sensitive information and it is not possible to convey it elsewhere, encapsulate the HTTP message using the application/http media type (Section 8.3.2 of [RFC7230]), encrypting that as the payload of the "outer" message.

4.7. Poisoning Storage

This mechanism only offers data origin authentication; it does not perform authentication or authorization of the message creator, which could still need to be performed (e.g., by HTTP authentication [RFC7235]).

This is especially relevant when a HTTP PUT request is accepted by a server without decrypting the payload; if the request is unauthenticated, it becomes possible for a third party to deny service and/or poison the store.

4.8. Sizing and Timing Attacks

Applications using this mechanism need to be aware that the size of encrypted messages, as well as their timing, HTTP methods, URIs and so on, may leak sensitive information. See for example [NETFLIX] or [CLINIC].

This risk can be mitigated through the use of the padding that this mechanism provides. Alternatively, splitting up content into segments and storing them separately might reduce exposure. HTTP/2 [RFC7540] combined with TLS [RFC5246] might be used to hide the size of individual messages.

Developing a padding strategy is difficult. A good padding strategy can depend on context. Common strategies include padding to a small set of fixed lengths, padding to multiples of a value, or padding to powers of 2. Even a good strategy can still cause size information to leak if processing activity of a recipient can be observed. This is especially true if the trailing records of a message contain only padding. Distributing non-padding data across records is recommended to avoid leaking size information.

5. IANA Considerations

5.1. The "aes128gcm" HTTP Content Coding

This memo registers the "aes128gcm" HTTP content coding in the HTTP Content Codings Registry, as detailed in Section 2.

- Name: aes128gcm
- Description: AES-GCM encryption with a 128-bit content encryption key
- Reference: this specification

6. References

6.1. Normative References

[FIPS180-4] National Institute of Standards and Technology, U.S. Department of Commerce, "NIST FIPS 180-4, Secure Hash Standard", DOI 10.6028/NIST.FIPS.180-4, August 2015, <http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf>.

[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>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

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

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <http://www.rfc-editor.org/info/rfc5869>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

6.2. Informative References

[AEBounds] Luykx, A. and K. Paterson, "Limits on Authenticated Encryption Use in TLS", March 2016, <http://www.isg.rhul.ac.uk/~kp/TLS-AEbounds.pdf>.

[CLINIC] Miller, B., Huang, L., Joseph, A., and J. Tygar, "I Know Why You Went to the Clinic: Risks and Realization of HTTPS Traffic Analysis", March 2014, <https://arxiv.org/abs/1403.0297>.

[NETFLIX] Reed, A. and M. Kranch, "Identifying HTTPS-Protected Netflix Videos in Real-Time", Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy - CODASPY '17 , DOI 10.1145/3029806.3029821, 2017.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <http://www.rfc-editor.org/info/rfc4648>.

[RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, DOI 10.17487/RFC4880, November 2007, <http://www.rfc-editor.org/info/rfc4880>.

[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>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <http://www.rfc-editor.org/info/rfc5652>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <http://www.rfc-editor.org/info/rfc7233>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <http://www.rfc-editor.org/info/rfc7235>.

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <http://www.rfc-editor.org/info/rfc7516>.

[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>.

[XMLENC] Eastlake, D., Reagle, J., Hirsch, F., Roessler, T., Imamura, T., Dillaway, B., Simon, E., Yiu, K., and M. Nystroem, "XML Encryption Syntax and Processing", W3C Recommendation REC-xmlenc-core1-20130411, January 2013, <https://www.w3.org/TR/2013/REC-xmlenc-core1-20130411>.

Appendix A. JWE Mapping

The "aes128gcm" content coding can be considered as a sequence of JSON Web Encryption (JWE) objects [RFC7516], each corresponding to a single fixed size record that includes trailing padding. The following transformations are applied to a JWE object that might be expressed using the JWE Compact Serialization:

- The JWE Protected Header is fixed to the value { "alg": "dir", "enc": "A128GCM" }, describing direct encryption using AES-GCM with a 128-bit content encryption key. This header is not transmitted, it is instead implied by the value of the Content-Encoding header field.

- The JWE Encrypted Key is empty, as stipulated by the direct encryption algorithm.

- The JWE Initialization Vector ("iv") for each record is set to the exclusive or of the 96-bit record sequence number, starting at zero, and a value derived from the input keying material (see Section 2.3). This value is also not transmitted.

- The final value is the concatenated header, JWE Ciphertext, and JWE Authentication Tag, all expressed without base64url encoding. The "." separator is omitted, since the length of these fields is known.

Thus, the example in Section 3.1 can be rendered using the JWE Compact Serialization as:

```
eyAiYWxnIjogImRpciIsICJlbmMiOiAiQTEyOEdDTSIgfQ..Bcs8gkIRKLI8GeI8.
-NAVub2qFgBEuQKRapoZuw.4jGQi9rcwQHU8P6XLxOGOA
```

Where the first line represents the fixed JWE Protected Header, an empty JWE Encrypted Key, and the algorithmically-determined JWE Initialization Vector. The second line contains the encoded body, split into JWE Ciphertext and JWE Authentication Tag.

Appendix B. Acknowledgements

Mark Nottingham was an original author of this document.

The following people provided valuable input: Richard Barnes, David Benjamin, Peter Beverloo, JR Conlin, Mike Jones, Stephen Farrell, Adam Langley, James Manger, John Mattsson, Julian Reschke, Eric Rescorla, Jim Schaad, and Magnus Westerlund.

Author’s Address

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

HTTP Working Group
Internet-Draft
Intended status: Experimental
Expires: September 18, 2017

M. Nottingham

M. Thomson
Mozilla

March 17, 2017

Opportunistic Security for HTTP/2
draft-ietf-httpbis-http2-encryption-11

Abstract

This document describes how "http" URIs can be accessed using Transport Layer Security (TLS) and HTTP/2 to mitigate pervasive monitoring attacks. This mechanism not a replacement for "https" URIs; it is vulnerable to active attacks.

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 18, 2017.

Table of Contents

1. Introduction 2
1.1. Goals and Non-Goals 2
1.2. Notational Conventions 3
2. Using HTTP URIs over TLS 3
2.1. Alternative Server Opt-In 4
2.2. Interaction with "https" URIs 5
2.3. The "http-opportunistic" well-known URI 5
3. IANA Considerations 6
4. Security Considerations 6
4.1. Security Indicators 6
4.2. Downgrade Attacks 6
4.3. Privacy Considerations 6
4.4. Confusion Regarding Request Scheme 7
4.5. Server Controls 7
5. References 7
5.1. Normative References 7
5.2. Informative References 8
Appendix A. Acknowledgements 9
Authors' Addresses 9

1. Introduction

This document describes a use of HTTP Alternative Services [RFC7838] to decouple the URI scheme from the use and configuration of underlying encryption. It allows an "http" URI to be accessed using HTTP/2 [RFC7230] and Transport Layer Security (TLS) [RFC5246] with Opportunistic Security [RFC7435].

This document describes a usage model whereby sites can serve "http" URIs over TLS, thereby avoiding the problem of serving Mixed Content (described in [W3C.CR-mixed-content-20160802]) while still providing protection against passive attacks.

Opportunistic Security does not provide the same guarantees as using TLS with "https" URIs, because it is vulnerable to active attacks, and does not change the security context of the connection. Normally, users will not be able to tell that it is in use (i.e., there will be no "lock icon").

1.1. Goals and Non-Goals

The immediate goal is to make the use of HTTP more robust in the face of pervasive passive monitoring [RFC7258].

A secondary (but significant) goal is to provide for ease of implementation, deployment and operation. This mechanism is expected

to have a minimal impact upon performance, and require a trivial administrative effort to configure.

Preventing active attacks (such as a Man-in-the-Middle) is a non-goal for this specification. Furthermore, this specification is not intended to replace or offer an alternative to "https", since "https" both prevents active attacks and invokes a more stringent security model in most clients.

1.2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Using HTTP URIs over TLS

An origin server that supports the resolution of "http" URIs can indicate support for this specification by providing an alternative service advertisement [RFC7838] for a protocol identifier that uses TLS, such as "h2" [RFC7540]. Such a protocol MUST include an explicit indication of the scheme of the resource. This excludes HTTP/1.1; HTTP/1.1 clients are forbidden from including the absolute form of a URI in requests to origin servers (see Section 5.3.1 of [RFC7230]).

A client that receives such an advertisement MAY make future requests intended for the associated origin [RFC6454] to the identified service (as specified by [RFC7838]), provided that the alternative service opts in as described in Section 2.1.

A client that places the importance of protection against passive attacks over performance might choose to withhold requests until an encrypted connection is available. However, if such a connection cannot be successfully established, the client can resume its use of the cleartext connection.

A client can also explicitly probe for an alternative service advertisement by sending a request that bears little or no sensitive information, such as one with the OPTIONS method. Likewise, clients with existing alternative services information could make such a request before they expire, in order minimize the delays that might be incurred.

Client certificates are not meaningful for URLs with the "http" scheme, and therefore clients creating new TLS connections to alternative services for the purposes of this specification MUST NOT present them. A server that also provides "https" resources on the

same port can request a certificate during the TLS handshake, but it MUST NOT abort the handshake if the client does not provide one.

2.1. Alternative Server Opt-In

It is possible that the server might become confused about whether requests' URLs have a "http" or "https" scheme, for various reasons; see Section 4.4. To ensure that the alternative service has opted into serving "http" URLs over TLS, clients are required to perform additional checks before directing "http" requests to it.

Clients MUST NOT send "http" requests over a secured connection, unless the chosen alternative service presents a certificate that is valid for the origin as defined in [RFC2818]. Using an authenticated alternative service establishes "reasonable assurances" for the purposes of [RFC7838]. In addition to authenticating the server, the client MUST have obtained a valid http-opportunistic response for an origin (as per Section 2.3) using the authenticated connection. An exception to the latter restriction is made for requests for the "http-opportunistic" well-known URI.

For example, assuming the following request is made over a TLS connection that is successfully authenticated for those origins, the following request/response pair would allow requests for the origins "http://www.example.com" or "http://example.com" to be sent using a secured connection:

```
HEADERS
  + END_STREAM
  + END_HEADERS
    :method = GET
    :scheme = http
    :authority = example.com
    :path = /.well-known/http-opportunistic

HEADERS
    :status = 200
    content-type = application/json
DATA
  + END_STREAM
[ "http://www.example.com", "http://example.com" ]
```

Though this document describes multiple origins, this is only for operational convenience. Only a request made to an origin (over an authenticated connection) can be used to acquire this resource for that origin. Thus in the example, the request to "http://example.com" cannot be assumed to also provide an http-opportunistic response for "http://www.example.com".

2.2. Interaction with "https" URIs

Clients MUST NOT send "http" requests and "https" requests on the same connection. Similarly, clients MUST NOT send "http" requests for multiple origins on the same connection.

2.3. The "http-opportunistic" well-known URI

This specification defines the "http-opportunistic" well-known URI [RFC5785]. A client is said to have a valid http-opportunistic response for a given origin when:

- The client has requested the well-known URI from the origin over an authenticated connection and a 200 (OK) response was provided, and
- That response is fresh [RFC7234] (potentially through revalidation [RFC7232]), and
- That response has the media type "application/json", and
- That response's payload, when parsed as JSON [RFC7159], contains an array as the root, and
- The array contains a string that is a case-insensitive character-for-character match for the origin in question, serialised into Unicode as per Section 6.1 of [RFC6454].

A client MAY treat an "http-opportunistic" resource as invalid if values it contains are not strings.

This document does not define semantics for "http-opportunistic" resources on an "https" origin, nor does it define semantics if the resource includes "https" origins.

Allowing clients to cache the http-opportunistic resource means that all alternative services need to be able to respond to requests for "http" resources. A client is permitted to use an alternative service without acquiring the http-opportunistic resource from that service.

A client MUST NOT use any cached copies of an http-opportunistic resource that was acquired (or revalidated) over an unauthenticated connection. To avoid potential errors, a client can request or revalidate the http-opportunistic resource before using any connection to an alternative service.

Clients that use cached http-opportunistic responses MUST ensure that their cache is cleared of any responses that were acquired over an unauthenticated connection. Revalidating an unauthenticated response using an authenticated connection does not ensure the integrity of the response.

3. IANA Considerations

This specification registers a Well-Known URI [RFC5785]:

- URI Suffix: http-opportunistic
- Change Controller: IETF
- Specification Document(s): Section 2.3 of [this specification]
- Related Information:

4. Security Considerations

4.1. Security Indicators

User Agents MUST NOT provide any special security indicators when an "http" resource is acquired using TLS. In particular, indicators that might suggest the same level of security as "https" MUST NOT be used (e.g., a "lock device").

4.2. Downgrade Attacks

A downgrade attack against the negotiation for TLS is possible.

For example, because the "Alt-Svc" header field [RFC7838] likely appears in an unauthenticated and unencrypted channel, it is subject to downgrade by network attackers. In its simplest form, an attacker that wants the connection to remain in the clear need only strip the "Alt-Svc" header field from responses.

4.3. Privacy Considerations

Cached alternative services can be used to track clients over time; e.g., using a user-specific hostname. Clearing the cache reduces the ability of servers to track clients; therefore clients MUST clear cached alternative service information when clearing other origin-based state (i.e., cookies).

4.4. Confusion Regarding Request Scheme

HTTP implementations and applications sometimes use ambient signals to determine if a request is for an "https" resource; for example, they might look for TLS on the stack, or a server port number of 443.

This might be due to expected limitations in the protocol (the most common HTTP/1.1 request form does not carry an explicit indication of the URI scheme and the resource might have been developed assuming HTTP/1.1), or it may be because how the server and application are implemented (often, they are two separate entities, with a variety of possible interfaces between them).

Any security decisions based upon this information could be misled by the deployment of this specification, because it violates the assumption that the use of TLS (or port 443) means that the client is accessing a HTTPS URI, and operating in the security context implied by HTTPS.

Therefore, server implementers and administrators need to carefully examine the use of such signals before deploying this specification.

4.5. Server Controls

This specification requires that a server send both an Alternative Service advertisement and host content in a well-known location to send HTTP requests over TLS. Servers SHOULD take suitable measures to ensure that the content of the well-known resource remains under their control. Likewise, because the Alt-Svc header field is used to describe policies across an entire origin, servers SHOULD NOT permit user content to set or modify the value of this header.

5. References

5.1. Normative References

[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>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

[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>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <http://www.rfc-editor.org/info/rfc5785>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <http://www.rfc-editor.org/info/rfc6454>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[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>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <http://www.rfc-editor.org/info/rfc7232>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <http://www.rfc-editor.org/info/rfc7234>.

[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>.

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

5.2. Informative References

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <http://www.rfc-editor.org/info/rfc7435>.

[RFC7469] Evans, C., Palmer, C., and R. Sleevi, "Public Key Pinning Extension for HTTP", RFC 7469, DOI 10.17487/RFC7469, April 2015, <http://www.rfc-editor.org/info/rfc7469>.

[W3C.CR-mixed-content-20160802] West, M., "Mixed Content", World Wide Web Consortium CR CR-mixed-content-20160802, August 2016, <https://www.w3.org/TR/2016/CR-mixed-content-20160802>.

Appendix A. Acknowledgements

Mike Bishop contributed significant text to this document.

Thanks to Patrick McManus, Stefan Eissing, Eliot Lear, Stephen Farrell, Guy Podjarny, Stephen Ludin, Erik Nygren, Paul Hoffman, Adam Langley, Eric Rescorla, Julian Reschke, Kari Hurtta, and Richard Barnes for their feedback and suggestions.

Authors' Addresses

Mark Nottingham

Email: mnot@mnot.net
URI: https://www.mnot.net/

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

HTTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 27, 2017

J. Reschke
greenbytes

October 24, 2016

A JSON Encoding for HTTP Header Field Values
draft-ietf-httpbis-jfv-02

Abstract

This document establishes a convention for use of JSON-encoded field values in HTTP header fields.

Editorial Note (To be removed by RFC Editor before publication)

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

Working Group information can be found at <http://httpwg.github.io/>; source code and issues list for this draft can be found at <https://github.com/httpwg/http-extensions>.

The changes in this draft are summarized in Appendix A.

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 April 27, 2017.

Copyright Notice

Table of Contents

1. Introduction 3
2. Data Model and Format 3
3. Sender Requirements 4
4. Recipient Requirements 5
5. Using this Format in Header Field Definitions 5
6. Examples 5
6.1. Content-Length 5
6.2. Content-Disposition 6
6.3. WWW-Authenticate 7
6.4. Accept-Encoding 8
7. Discussion 9
8. Deployment Considerations 9
9. Interoperability Considerations 9
9.1. Encoding and Characters 9
9.2. Numbers 10
9.3. Object Constraints 10
10. Internationalization Considerations 10
11. Security Considerations 10
12. References 11
12.1. Normative References 11
12.2. Informative References 11
Appendix A. Change Log (to be removed by RFC Editor before publication) 12
A.1. Since draft-reschke-http-jfv-00 12
A.2. Since draft-reschke-http-jfv-01 12
A.3. Since draft-reschke-http-jfv-02 12
A.4. Since draft-reschke-http-jfv-03 13
A.5. Since draft-reschke-http-jfv-04 13
A.6. Since draft-ietf-httpbis-jfv-00 13
A.7. Since draft-ietf-httpbis-jfv-01 13
Appendix B. Acknowledgements 13

1. Introduction

Defining syntax for new HTTP header fields ([RFC7230], Section 3.2) is non-trivial. Among the commonly encountered problems are:

- There is no common syntax for complex field values. Several well-known header fields do use a similarly looking syntax, but it is hard to write generic parsing code that will both correctly handle valid field values but also reject invalid ones.

- The HTTP message format allows header fields to repeat, so field syntax needs to be designed in a way that these cases are either meaningful, or can be unambiguously detected and rejected.

- HTTP/1.1 does not define a character encoding scheme ([RFC6365], Section 2), so header fields are either stuck with US-ASCII ([RFC0020]), or need out-of-band information to decide what encoding scheme is used. Furthermore, APIs usually assume a default encoding scheme in order to map from octet sequences to strings (for instance, [XMLHttpRequest] uses the IDL type "ByteString", effectively resulting in the ISO-8859-1 character encoding scheme [ISO-8859-1] being used).

(See Section 8.3.1 of [RFC7231] for a summary of considerations for new header fields.)

This specification addresses the issues listed above by defining both a generic JSON-based ([RFC7159]) data model and a concrete wire format that can be used in definitions of new header fields, where the goals were:

- to be compatible with header field recombination when fields occur multiple times in a single message (Section 3.2.2 of [RFC7230]), and

- not to use any problematic characters in the field value (non-ASCII characters and certain whitespace characters).

2. Data Model and Format

In HTTP, header fields with the same field name can occur multiple times within a single message (Section 3.2.2 of [RFC7230]). When this happens, recipients are allowed to combine the field values using commas as delimiter. This rule matches nicely JSON's array format (Section 5 of [RFC7159]). Thus, the basic data model used here is the JSON array.

Header field definitions that need only a single value can restrict

themselves to arrays of length 1, and are encouraged to define error handling in case more values are received (such as "first wins", "last wins", or "abort with fatal error message").

JSON arrays are mapped to field values by creating a sequence of serialized member elements, separated by commas and optionally whitespace. This is equivalent to using the full JSON array format, while leaving out the "begin-array" ('[') and "end-array" (']') delimiters.

The ABNF character names and classes below are used (copied from [RFC5234], Appendix B.1):

```
  CR              = %x0D    ; carriage return
  HTAB            = %x09    ; horizontal tab
  LF              = %x0A    ; line feed
  SP              = %x20    ; space
  VCHAR           = %x21-7E ; visible (printing) characters
```

Characters in JSON strings that are not allowed or discouraged in HTTP header field values -- that is, not in the "VCHAR" definition -- need to be represented using JSON's "backslash" escaping mechanism ([RFC7159], Section 7).

The control characters CR, LF, and HTAB do not appear inside JSON strings, but can be used outside (line breaks, indentation etc.). These characters need to be either stripped or replaced by space characters (ABNF "SP").

Formally, using the HTTP specification's ABNF extensions defined in Section 7 of [RFC7230]:

```
  json-field-value = #json-field-item
  json-field-item  = JSON-Text
                   ; see [RFC7159], Section 2,
                   ; post-processed so that only VCHAR characters
                   ; are used
```

3. Sender Requirements

To map a JSON array to an HTTP header field value, process each array element separately by:

1. generating the JSON representation,

2. stripping all JSON control characters (CR, HTAB, LF), or replacing them by space ("SP") characters,

3. replacing all remaining non-VSPACE characters by the equivalent backslash-escape sequence ([RFC7159], Section 7).

The resulting list of strings is transformed into an HTTP field value by combining them using comma (%x2C) plus optional SP as delimiter, and encoding the resulting string into an octet sequence using the US-ASCII character encoding scheme ([RFC0020]).

4. Recipient Requirements

To map a set of HTTP header field instances to a JSON array:

1. combine all header field instances into a single field as per Section 3.2.2 of [RFC7230],

2. add a leading begin-array ("[") octet and a trailing end-array ("]") octet, then

3. run the resulting octet sequence through a JSON parser.

The result of the parsing operation is either an error (in which case the header field values needs to be considered invalid), or a JSON array.

5. Using this Format in Header Field Definitions

[[anchor5: Explain what a definition of a new header field needs to do precisely to use this format, mention must-ignore extensibility]]

6. Examples

This section shows how some of the existing HTTP header fields would look like if they would use the format defined by this specification.

6.1. Content-Length

"Content-Length" is defined in Section 3.3.2 of [RFC7230], with the field value’s ABNF being:

```
Content-Length = 1*DIGIT
```

So the field value is similar to a JSON number ([RFC7159], Section 6).

Content-Length is restricted to a single field instance, as it doesn’t use the list production (as per Section 3.2.2 of [RFC7230]). However, in practice multiple instances do occur, and the definition of the header field does indeed discuss how to handle these cases.

If Content-Length was defined using the JSON format discussed here, the ABNF would be something like:

```
Content-Length = #number
                 ; number: [RFC7159], Section 6
```

...and the prose definition would:

- restrict all numbers to be non-negative integers without fractions, and
- require that the array of values is of length 1 (but allow the case where the array is longer, but all members represent the same value)

6.2. Content-Disposition

Content-Disposition field values, defined in [RFC6266], consist of a "disposition type" (a string), plus multiple parameters, of which at least one ("filename") sometime needs to carry non-ASCII characters.

For instance, the first example in Section 5 of [RFC6266]:

```
Attachment; filename=example.html
```

has a disposition type of "Attachment", with filename parameter value "example.html". A JSON representation of this information might be:

```
{
  "Attachment": {
    "filename" : "example.html"
  }
}
```

which would translate to a header field value of:

```
{ "Attachment": { "filename" : "example.html" } }
```

The third example in Section 5 of [RFC6266] uses a filename parameter containing non-US-ASCII characters:

```
attachment; filename*=UTF-8''%e2%82%ac%20rates
```

Note that in this case, the "filename*" parameter uses the encoding defined in [RFC5987], representing a filename starting with the Unicode character U+20AC (EURO SIGN), followed by " rates". If the definition of Content-Disposition would have used the format proposed here, the workaround involving the "parameter*" syntax would not have

been needed at all.

The JSON representation of this value could then be:

```
{ "attachment": { "filename" : "\u20AC rates" } }
```

6.3. WWW-Authenticate

The WWW-Authenticate header field value is defined in Section 4.1 of [RFC7235] as a list of "challenges":

```
WWW-Authenticate = 1#challenge
```

...where a challenge consists of a scheme with optional parameters:

```
challenge   = auth-scheme [ 1*SP ( token68 / #auth-param ) ]
```

An example for a complex header field value given in the definition of the header field is:

```
Newauth realm="apps", type=1, title="Login to \"apps\"",
Basic realm="simple"
```

(line break added for readability)

A possible JSON representation of this field value would be the array below:

```
[
  {
    "Newauth" : {
      "realm": "apps",
      "type" : 1,
      "title" : "Login to \"apps\""
    }
  },
  {
    "Basic" : {
      "realm": "simple"
    }
  }
]
```

...which would translate to a header field value of:

```
{ "Newauth" : { "realm": "apps", "type" : 1,
                "title": "Login to \"apps\"" }},
{ "Basic" : { "realm": "simple"}}
```

6.4. Accept-Encoding

The Accept-Encoding header field value is defined in Section 5.3.4 of [RFC7231] as a list of codings, each of which allowing a weight parameter 'q':

```
Accept-Encoding = #( codings [ weight ] )
codings         = content-coding / "identity" / "*"
weight          = OWS ";" OWS "q=" qvalue
qvalue          = ( "0" [ "." 0*3DIGIT ] )
                / ( "1" [ "." 0*3("0") ] )
```

An example for a complex header field value given in the definition of the header field is:

```
gzip;q=1.0, identity; q=0.5, *;q=0
```

Due to the defaulting rules for the quality value ([RFC7231], Section 5.3.1), this could also be written as:

```
gzip, identity; q=0.5, *; q=0
```

A JSON representation could be:

```
[
  {
    "gzip" : {
    }
  },
  {
    "identity" : {
      "q": 0.5
    }
  },
  {
    "*" : {
      "q": 0
    }
  }
]
```

...which would translate to a header field value of:

```
{"gzip": {}}, {"identity": {"q": 0.5}}, {"*": {"q": 0}}
```

In this example, the part about "gzip" appears unnecessarily verbose, as the value is just an empty object. A simpler notation would collapse members like these to string literals:

```
  "gzip", {"identity": {"q": 0.5}}, {"*": {"q": 0}}
```

If this is desirable, the header field definition could allow both string literals and objects, and define that a mere string literal would be mapped to a member whose name is given by the string literal, and the value is an empty object.

For what it's worth, one of the most common cases for 'Accept-Encoding' would become:

```
  "gzip", "deflate"
```

which would be only a small overhead over the original format.

7. Discussion

This approach uses a default of "JSON array", using implicit array markers. An alternative would be a default of "JSON object". This would simplify the syntax for non-list-typed header fields, but all the benefits of having the same data model for both types of header fields would be gone. A hybrid approach might make sense, as long as it doesn't require any heuristics on the recipient's side.

> Note: a concrete proposal was made by Kazuho Oku in <https://lists.w3.org/Archives/Public/ietf-http-wg/2016JanMar/0155.html>.

[[anchor7: Use of generic libs vs compactness of field values..]]

[[anchor8: Mention potential "Key" header field extension ([KEY]).]]

8. Deployment Considerations

This JSON-based syntax will only apply to newly introduced header fields, thus backwards compatibility is not a problem. That being said, it is conceivable that there is existing code that might trip over double quotes not being used for HTTP's quoted-string syntax (Section 3.2.6 of [RFC7230]).

9. Interoperability Considerations

The "I-JSON Message Format" specification ([RFC7493]) addresses known JSON interoperability pain points. This specification borrows from the requirements made over there:

9.1. Encoding and Characters

This specification requires that field values use only US-ASCII characters, and thus by definition use a subset of UTF-8 (Section 2.1

of [RFC7493]).

9.2. Numbers

Be aware of the issues around number precision, as discussed in Section 2.2 of [RFC7493].

9.3. Object Constraints

As described in Section 4 of [RFC7159], JSON parser implementations differ in the handling of duplicate object names. Therefore, senders MUST NOT use duplicate object names, and recipients SHOULD either treat field values with duplicate names as invalid (consistent with [RFC7493], Section 2.3) or use the lexically last value (consistent with [ECMA-262], Section 24.3.1.1).

Furthermore, ordering of object members is not significant and can not be relied upon.

10. Internationalization Considerations

In HTTP/1.1, header field values are represented by octet sequences, usually used to transmit ASCII characters, with restrictions on the use of certain control characters, and no associated default character encoding, nor a way to describe it ([RFC7230], Section 3.2). HTTP/2 does not change this.

This specification maps all characters which can cause problems to JSON escape sequences, thereby solving the HTTP header field internationalization problem.

Future specifications of HTTP might change to allow non-ASCII characters natively. In that case, header fields using the syntax defined by this specification would have a simple migration path (by just stopping to require escaping of non-ASCII characters).

11. Security Considerations

Using JSON-shaped field values is believed to not introduce any new threads beyond those described in Section 12 of [RFC7159], namely the risk of recipients using the wrong tools to parse them.

Other than that, any syntax that makes extensions easy can be used to smuggle information through field values; however, this concern is shared with other widely used formats, such as those using parameters in the form of name/value pairs.

12. References

12.1. Normative References

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <http://www.rfc-editor.org/info/rfc20>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <http://www.rfc-editor.org/info/rfc7493>.

12.2. Informative References

[ECMA-262] Ecma International, "ECMA-262 6th Edition, The ECMAScript 2015 Language Specification", Standard ECMA-262, June 2015, <http://www.ecma-international.org/ecma-262/6.0/>.

[ISO-8859-1] International Organization for Standardization, "Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1", ISO/IEC 8859-1:1998, 1998.

[KEY] Fielding, R. and M. Nottingham, "The Key HTTP Response Header Field", draft-ietf-httpbis-key-01 (work in progress), March 2016.

[RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", RFC 5987, DOI 10.17487/RFC5987, August 2010, <http://www.rfc-editor.org/info/rfc5987>.

[RFC6266] Reschke, J., "Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP)", RFC 6266, DOI 10.17487/RFC6266, June 2011, <http://www.rfc-editor.org/info/rfc6266>.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <http://www.rfc-editor.org/info/rfc6365>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <http://www.rfc-editor.org/info/rfc7235>.

[XMLHttpRequest] van Kesteren, A., Aubourg, J., Song, J., and H. Steen, "XMLHttpRequest Level 1", W3C Working Draft WD-XMLHttpRequest-20140130, January 2014, <http://www.w3.org/TR/2014/WD-XMLHttpRequest-20140130/>.

Latest version available at <http://www.w3.org/TR/XMLHttpRequest/>.

Appendix A. Change Log (to be removed by RFC Editor before publication)

A.1. Since draft-reschke-http-jfv-00

Editorial fixes + working on the TODOs.

A.2. Since draft-reschke-http-jfv-01

Mention slightly increased risk of smuggling information in header field values.

A.3. Since draft-reschke-http-jfv-02

Mention Kazuho Oku’s proposal for abbreviated forms.

Added a bit of text about the motivation for a concrete JSON subset (ack Cory Benfield).

Expand I18N section.

A.4. Since draft-reschke-http-jfv-03

Mention relation to KEY header field.

A.5. Since draft-reschke-http-jfv-04

Change to HTTP Working Group draft.

A.6. Since draft-ietf-httpbis-jfv-00

Added example for "Accept-Encoding" (inspired by Kazuho's feedback), showing a potential way to optimize the format when default values apply.

A.7. Since draft-ietf-httpbis-jfv-01

Add interop discussion, building on I-JSON and ECMA-262 (see <https://github.com/httpwg/http-extensions/issues/225>).

Appendix B. Acknowledgements

Thanks go to the Hypertext Transfer Protocol Working Group participants.

Author's Address

Julian F. Reschke
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: http://greenbytes.de/tech/webdav/

HTTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 3, 2016

R. Fielding
Adobe Systems Incorporated
M. Nottingham

March 2, 2016

The Key HTTP Response Header Field
draft-ietf-httpbis-key-01

Abstract

The 'Key' header field for HTTP responses allows an origin server to describe the secondary cache key (RFC 7234, Section 4.1) for a resource, by conveying what is effectively a short algorithm that can be used upon later requests to determine if a stored response is reusable for a given request.

Key has the advantage of avoiding an additional round trip for validation whenever a new request differs slightly, but not significantly, from prior requests.

Key also informs user agents of the request characteristics that might result in different content, which can be useful if the user agent is not sending request header fields in order to reduce the risk of fingerprinting.

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/ .

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

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 3, 2016.

Table of Contents

1. Introduction 3
1.1. Examples 3
1.2. Notational Conventions 4
2. The "Key" Response Header Field 4
2.1. Relationship with Vary 5
2.2. Calculating a Secondary Cache Key 6
2.2.1. Creating a Header Field Value 8
2.2.2. Failing Parameter Processing 8
2.3. Key Parameters 9
2.3.1. div 9
2.3.2. partition 10
2.3.3. match 11
2.3.4. substr 12
2.3.5. param 13
3. IANA Considerations 14
3.1. Procedure 14
3.2. Registrations 14
4. Security Considerations 15
5. References 15
5.1. Normative References 15
5.2. Informative References 16
Appendix A. Acknowledgements 16
Appendix B. Changes 16
B.1. Since -00 16
Authors' Addresses 16

1. Introduction

In HTTP caching [RFC7234], the Vary response header field effectively modifies the key used to store and access a response to include information from the request's headers. This "secondary cache key" allows proactive content negotiation [RFC7231] to work with caches.

Vary's operation is generic; it works well when caches understand the semantics of the selecting headers. For example, the Accept-Language request header field has a well-defined syntax for expressing the client's preferences; a cache that understands this header field can select the appropriate response (based upon its Content-Language header field) and serve it to a client, without any knowledge of the underlying resource.

Vary does not work as well when the criteria for selecting a response are specific to the resource. For example, if the nature of the response depends upon the presence or absence of a particular Cookie ([RFC6265]) in a request, Vary doesn't have a mechanism to offer enough fine-grained, resource-specific information to aid a cache's selection of the appropriate response.

This document defines a new response header field, "Key", that allows resources to describe the secondary cache key in a fine-grained, resource-specific manner, leading to improved cache efficiency when responses depend upon such headers.

1.1. Examples

For example, this response header field:

```
Key: cookie;param=_sess;param=ID
```

indicates that the selected response depends upon the "_sess" and "ID" cookie values.

This Key:

```
Key: user-agent;substr=MSIE
```

indicates that there are two possible secondary cache keys for this resource; one for requests whose User-Agent header field contains "MSIE", and another for those that don't.

A more complex example:

```
Key: user-agent;substr=MSIE;Substr="mobile", Cookie;param="ID"
```

indicates that the selected response depends on the presence of two strings in the User-Agent request header field, as well as the value of the "ID" cookie request header field.

1.2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] (including the DQUOTE rule), and the list rule extension defined in [RFC7230], Section 7. It includes by reference the field-name, quoted-string and quoted-pair rules from that document, the OWS rule from [RFC7230] and the parameter rule from [RFC7231].

2. The "Key" Response Header Field

The "Key" response header field describes the portions of the request that the resource currently uses to select representations.

As such, its semantics are similar to the "Vary" response header field, but it allows more fine-grained description, using "key parameters".

Caches can use this information as part of determining whether a stored response can be used to satisfy a given request. When a cache knows and fully understands the Key header field for a given resource, it MAY ignore the Vary response header field in any stored responses for it.

Additionally, user agents can use Key to discover if additional request header fields might influence the resource’s selection of responses.

The Key field-value is a comma-delimited list of selecting header fields (similar to Vary), with zero to many parameters each, delimited by semicolons.

```
Key       = 1#key-value
key-value = field-name *( OWS ";" OWS parameter )
```

Note that, as per [RFC7231], parameter names are case-insensitive, and parameter values can be double-quoted strings (potentially with "\"-escaped characters inside).

The following header fields have the same effect:

```
Vary: Accept-Encoding, Cookie
Key: Accept-Encoding, Cookie
```

However, Key's use of parameters allows:

```
Key: Accept-Encoding, Cookie; param=foo
```

to indicate that the secondary cache key depends upon the Accept-Encoding header field and the "foo" Cookie.

One important difference between Vary and Key is how they are applied. Vary is specified to be specific to the response it occurs within, whereas Key is specific to the resource (as identified by the request URL) it is associated with. The most recent key you receive for a given resource is applicable to all responses from that resource.

This difference allows more efficient implementation (and reflects practices that many caches use in implementing Vary already).

This specification defines a selection of Key parameters to address common use cases such as selection upon individual Cookie header fields, User-Agent substrings and numerical ranges. Future parameters may define further capabilities.

2.1. Relationship with Vary

Origin servers SHOULD still send Vary when using Key, to ensure backwards compatibility.

For example,

```
Vary: User-Agent
Key: User-Agent;substr="mozilla"
```

Note that, in some cases, it may be better to explicitly use "Vary: *" if clients and caches don't have any practical way to use the Vary header field's value. For example,

```
Vary: *
Key: Cookie;param="ID"
```

Except when Vary: * is used, the set of headers used in Key SHOULD reflect the same request header fields as Vary does, even if they don't have parameters. For example,

```
Vary: Accept-Encoding, User-Agent
Key: Accept-Encoding, User-Agent;substr="mozilla"
```

Here, Accept-Encoding is included in Key without parameters; caches MAY treat these as they do values in the Vary header, relying upon knowledge of their generic semantics to select an appropriate response.

2.2. Calculating a Secondary Cache Key

When used by a cache to determine whether a stored response can be used to satisfy a presented request, each field-name in Key identifies a potential request header, just as with the Vary response header field.

However, each of these can have zero to many key parameters that change how the response selection process (as defined in [RFC7234], Section 4.3)) works.

In particular, when a cache fully implements this specification, it creates a secondary cache key for every request by following the instructions in the Key header field, ignoring the Vary header for this purpose.

Then, when a new request is presented, the secondary cache key generated for that request can be compared to the stored one to find the appropriate response, to determine if it can be selected.

To generate a secondary cache key for a given request (including that which is stored with a response) using Key, the following steps are taken:

1) If the Key header field is not present on the most recent cacheable (as per [RFC7234], Section 3)) response seen for the resource, abort this algorithm (i.e., fall back to using Vary to determine the secondary cache key).
2) Let "key_value" be the result of Creating a Header Field Value (Section 2.2.1) with "key" as the "target_field_name" and the most recently seen response header list for the resource as "header_list".
3) Let "secondary_key" be an empty string.

4) Create "key_list" by splitting "key_value" on "," characters, excepting "," characters within quoted strings, as per [RFC7230] Section 3.2.6..
5) For "key_item" in "key_list":

    1) Remove any leading and trailing WSP from "key_item".
    2) If "key_item" does not contain a ";" character, fail parameter processing (Section 2.2.2) and skip to the next "key_item".
    3) Let "field_name" be the string before the first ";" character in "key_item", removing any WSP between them.
    4) Let "field_value" be the result of Creating a Header Field Value (Section 2.2.1) with "field_name" as the "target_field_name" and the request header list as "header_list".
    5) Let "parameters" be the string after the first ";" character in "key_item", removing any WSP between them.
    6) Create "param_list" by splitting "parameters" on ";" characters, excepting ";" characters within quoted strings, as per [RFC7230] Section 3.2.6.
    7) For "parameter" in "param_list":

        1) If "parameter" does not contain a "=", fail parameter processing (Section 2.2.2) and skip to the next "key_item".
        2) Remove any WSP at the beginning and/or end of "parameter".
        3) Let "param_name" be the string before the first "=" character in "parameter", case-normalized to lowercase.
        4) If "param_name" does not identify a Key parameter processing algorithm that is implemented, fail parameter processing (Section 2.2.2) and skip to the next "key_item".
        5) Let "param_value" be the string after the first "=" character in "parameter".
        6) If the first and last characters of "param_value" are both DQUOTE:

            1) Remove the first and last characters of "param_value".
            2) Replace quoted-pairs within "param_value" with the octet following the backslash, as per [RFC7230] Section 3.2.6.
        7) If "param_value" does not conform to the syntax defined for it by the parameter definition, fail parameter processing Section 2.2.2 and skip to the next "key_item".
        8) Run the identified processing algorithm on "field_value" with the "param_value", and append the result to

"secondary_key". If parameter processing fails Section 2.2.2, skip to the next "key_item".
9) Append a separator character (e.g., NULL) to "secondary_key".
6) Return "secondary_key".

Note that this specification does not require that exact algorithm to be implemented. However, implementations' observable behavior MUST be identical to running it. This includes parameter processing algorithms; implementations MAY use different internal artefacts for secondary cache keys, as long as the results are the same.

Likewise, while the secondary cache key associated with both stored and presented requests is required to use the most recently seen Key header field for the resource in question, this can be achieved using a variety of implementation strategies, including (but not limited to):

- Generating a new secondary cache key for every stored response associated with the resource upon each request.
- Caching the secondary cache key with the stored request/response pair and re-generating it when the Key header field is observed to change.
- Caching the secondary cache key with the stored response and invalidating the stored response(s) when the Key header field is observed to change.

2.2.1. Creating a Header Field Value

Given a header field name "target_field_name" and "header_list", a list of ("field_name", "field_value") tuples:

1) Let "target_field_values" be an empty list.
2) For each ("field_name", "field_value") tuple in "header_list":
    1) If "field_name" does not match "target_field_name", skip to the next tuple.
    2) Strip leading and trailing WSP from "field_value" and append it to "target_field_values".
3) If "target_field_values" is empty, return an empty string.
4) Return the concatenation of "target_field_values", separating each with "," characters.

2.2.2. Failing Parameter Processing

In some cases, a key parameter cannot determine a secondary cache key corresponding to its nominated header field value. When this

happens, Key processing needs to fail safely, so that the correct behavior is observed.

When this happens, implementations MUST either behave as if the Key header was not present, or assure that the nominated header fields being compared match, as per [RFC7234], Section 4.1.

2.3. Key Parameters

A Key parameter associates a name with a specific processing algorithm that takes two inputs; a HTTP header value "header_value" (as described in Section 2.2.1), and "parameter_value", a string that indicates how the identified header should be processed.

The set of key parameters (and their associated processing algorithms) is extensible; see Section 3. This document defines the following key parameters:

2.3.1. div

The "div" parameter normalizes positive integer header values into groups by dividing them by a configured value.

Its value’s syntax is:

```
div    = 1*DIGIT
```

To process a set of header fields against a div parameter, follow these steps (or their equivalent):

1) If "parameter_value" is "0", fail parameter processing Section 2.2.2.
2) If "header_value" is the empty string, return "none".
3) If "header_value" contains a ",", remove it and all subsequent characters.
4) Remove all WSP characters from "header_value".
5) If "header_value" does not match the div ABNF rule, fail parameter processing (Section 2.2.2).
6) Return the quotient of "header_value" / "parameter_value" (omitting the modulus).

For example, the Key:

```
Key: Bar;div=5
```

indicates that the "Bar" header's field value should be partitioned into groups of 5. Thus, the following field values would be considered the same (because, divided by 5, they all result in 0):

```
Bar: 1
Bar: 3 , 42
Bar: 4, 1
```

whereas these would be considered to be in a different group (because, divided by 5, they all result in 2);

```
Bar: 12
Bar: 10
Bar: 14, 1
```

2.3.2. partition

The "partition" parameter normalizes positive numeric header values into pre-defined segments.

Its value's syntax is:

```
partition = [ segment ] *( ":" [ segment ] )
segment   = [ 0*DIGIT "." ] 1*DIGIT
```

To process a set of header fields against a partition parameter, follow these steps (or their equivalent):

1) If "header_value" is the empty string, return "none".
2) If "header_value" contains a ",", remove it and all subsequent characters.
3) Remove all WSP characters from "header_value".
4) If "header_value" does not match the segment ABNF rule, fail parameter processing (Section 2.2.2).
5) Let "segment_id" be 0.
6) Create a list "segment_list" by splitting "parameter_value" on ":" characters.
7) For each "segment_value" in "segment_list":
    1) If "header_value" is less than "segment_value" when they are numerically compared, skip to step 7.
    2) Increment "segment_id" by 1.
8) Return "segment_id".

For example, the Key:

Key: Foo;partition=20:30:40

indicates that the "Foo" header's field value should be divided into four segments:

- less than 20
- 20 to less than 30
- 30 to less than 40
- forty or greater

Thus, the following headers would all be normalized to the first segment:

```
Foo: 1
Foo: 0
Foo: 4, 54
Foo: 19.9
```

whereas the following would fall into the second segment:

```
Foo: 20
Foo: 29.999
Foo:  24   , 10
```

2.3.3. match

The "match" parameter is used to determine if an exact value occurs in a list of header values. It is case-sensitive.

Its value's syntax is:

```
match  = ( token / quoted-string )
```

To process a set of header fields against a match parameter, follow these steps (or their equivalent):

1) If "header_value" is the empty string, return "none".
2) Create "header_list" by splitting "header_value" on "," characters.
3) For each "header_item" in "header_list":
    1) Remove leading and trailing WSP characters in "header_item".
    2) If the value of "header_item" is character-for-character identical to "parameter_value", return "1".
4) Return "0".

For example, the Key:

```
Key: Baz;match="charlie"
```

Would return "1" for the following header field values:

```
Baz: charlie
Baz: foo, charlie
Baz: bar, charlie     , abc
```

and "0" for these:

```
Baz: theodore
Baz: joe, sam
Baz: "charlie"
Baz: Charlie
Baz: cha rlie
Baz: charlie2
```

2.3.4. substr

The "substr" parameter is used to determine if a value occurs as a substring of an item in a list of header values. It is case-sensitive.

Its value’s syntax is:

```
substr  = ( token / quoted-string )
```

To process a set of header fields against a substr parameter, follow these steps (or their equivalent):

1) If "header_value" is the empty string, return "none".
2) Create "header_list" by splitting "header_value" on "," characters.
3) For each "header_item" in "header_list":
    1) Remove leading and trailing WSP characters in "header_item".
    2) If the value of "parameter_value" is character-for-character present as a substring of "header_value", return "1".
4) Return "0".

For example, the Key:

Key: Abc;substr=bennet

Would return "1" for the following header field values:

```
Abc: bennet
Abc: foo, bennet
Abc: abennet00
Abc: bar, 99bennet     , abc
Abc: "bennet"
```

and "0" for these:

```
Abc: theodore
Abc: joe, sam
Abc: Bennet
Abc: Ben net
```

2.3.5. param

The "param" parameter considers the request header field as a list of key=value parameters, and uses the nominated key's value as the secondary cache key.

Its value's syntax is:

```
param  = ( token / quoted-string )
```

To process a list of header fields against a param parameter, follow these steps (or their equivalent):

1) Let "header_list" be an empty list.
2) Create "header_list_tmp1" by splitting header_value on "," characters.
3) For each "header_item_tmp1" in "header_list_tmp1":
    1) Create "header_list_tmp2" by splitting "header_item_tmp1" on ";" characters.
    2) For each "header_item_tmp2" in "header_list_tmp2":
        1) Remove leading and trailing WSP from "header_item_tmp2".
        2) Append "header_item_tmp2" to header_list.
4) For each "header_item" in "header_list":
    1) If the "=" character does not occur within "header_item", skip to the next "header_item".

2) Let "item_name" be the string occurring before the first "=" character in "header_item".
3) If "item_name" does not case-insensitively match "parameter_value", skip to the next "header_item".
4) Return the string occurring after the first "=" character in "header_item".

5) Return the empty string.

Note that steps 2 and 3 accommodate semicolon-separated values, so that it can be used with the Cookie request header field.

For example, the Key:

```
Key: Def;param=liam
```

The following headers would return the string (surrounded in single quotes) indicated:

```
Def: liam=123           // '123'
Def: mno=456            // ''
Def:                    // ''
Def: abc=123; liam=890  // '890'
Def: liam="678"         // '"678"'
```

3. IANA Considerations

This specification defines the HTTP Key Parameter Registry, maintained at http://www.iana.org/assignments/http-parameters/http-parameters.xhtml#key .

3.1. Procedure

Key Parameter registrations MUST include the following fields:

- Parameter Name: [name]
- Reference: [Pointer to specification text]

Values to be added to this namespace require IETF Review (see Section 4.1 of [RFC5226]) and MUST conform to the purpose of content coding defined in this section.

3.2. Registrations

This specification makes the following entries in the HTTP Key Parameter Registry:

Parameter Name	Reference
div	Section 2.3.1
partition	Section 2.3.2
match	Section 2.3.3
substr	Section 2.3.4
param	Section 2.3.5

4. Security Considerations

Because Key is an alternative to Vary, it is possible for caches to behave differently based upon whether they implement Key. Likewise, because support for any one Key parameter is not required, it is possible for different implementations of Key to behave differently. In both cases, an attacker might be able to exploit these differences.

This risk is mitigated by the requirement to fall back to Vary when unsupported parameters are encountered, coupled with the requirement that servers that use Key also include a relevant Vary header.

An attacker with the ability to inject response headers might be able to perform a cache poisoning attack that tailors a response to a specific user (e.g., by Keying to a Cookie that’s specific to them). While the attack is still possible without Key, the ability to tailor is new.

When implemented, Key might result in a larger number of stored responses for a given resource in caches; this, in turn, might be used to create an attack upon the cache itself. Good cache replacement algorithms and denial of service monitoring in cache implementations are reasonable mitigations against this risk.

5. References

5.1. Normative References

[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>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <http://www.rfc-editor.org/info/rfc7234>.

5.2. Informative References

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <http://www.rfc-editor.org/info/rfc6265>.

Appendix A. Acknowledgements

Thanks to Ilya Grigorik, Amos Jeffries and Yoav Weiss for their feedback.

Appendix B. Changes

B.1. Since -00

- Issue 108 (field-name cardinality) closed with no action.
- Issue 104 (Support "Or" operator) closed with no action.
- Issue 107 (Whitespace requirement) addressed by allowing whitespace around parameters.
- Issue 106 (Policy for Key parameter registry) closed with no action.

Authors' Addresses

Roy T. Fielding
Adobe Systems Incorporated

Email: fielding@gbiv.com
URI: http://roy.gbiv.com/

Mark Nottingham

Email: mnot@mnot.net
URI: https://www.mnot.net/

HTTP
Internet-Draft
Intended status: Standards Track
Expires: July 17, 2018

M. Nottingham

E. Nygren
Akamai

January 13, 2018

The ORIGIN HTTP/2 Frame
draft-ietf-httpbis-origin-frame-06

Abstract

This document specifies the ORIGIN frame for HTTP/2, to indicate what origins are available on a given connection.

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/origin-frame [3].

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 July 17, 2018.

Table of Contents

1. Introduction 2
1.1. Notational Conventions 3
2. The ORIGIN HTTP/2 Frame 3
2.1. Syntax 3
2.2. Processing ORIGIN Frames 4
2.3. The Origin Set 5
2.4. Authority, Push and Coalescing with ORIGIN 6
3. IANA Considerations 7
4. Security Considerations 7
5. References 7
5.1. Normative References 7
5.2. Informative References 8
5.3. URIs 9
Appendix A. Non-Normative Processing Algorithm 9
Appendix B. Operational Considerations for Servers 9
Authors' Addresses 10

1. Introduction

HTTP/2 [RFC7540] allows clients to coalesce different origins [RFC6454] onto the same connection when certain conditions are met. However, in certain cases, a connection is not usable for a coalesced origin, so the 421 (Misdirected Request) status code ([RFC7540], Section 9.1.2) was defined.

Using a status code in this manner allows clients to recover from misdirected requests, but at the penalty of adding latency. To address that, this specification defines a new HTTP/2 frame type, "ORIGIN", to allow servers to indicate what origins a connection is usable for.

Additionally, experience has shown that HTTP/2's requirement to establish server authority using both DNS and the server's certificate is onerous. This specification relaxes the requirement to check DNS when the ORIGIN frame is in use. Doing so has

additional benefits, such as removing the latency associated with some DNS lookups.

1.1. Notational Conventions

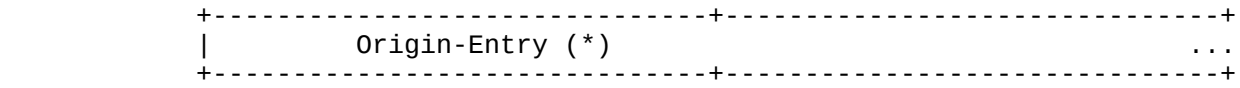
The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. The ORIGIN HTTP/2 Frame

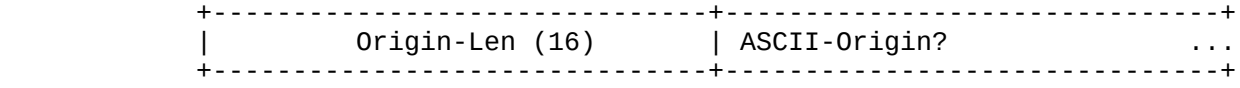
This document defines a new HTTP/2 frame type ([RFC7540], Section 4) called ORIGIN, that allows a server to indicate what origin(s) [RFC6454] the server would like the client to consider as members of the Origin Set (Section 2.3) for the connection it occurs within.

2.1. Syntax

The ORIGIN frame type is 0xc (decimal 12), and contains zero or more instances of the Origin-Entry field.



An Origin-Entry is a length-delimited string:



Specifically:

Origin-Len: An unsigned, 16-bit integer indicating the length, in octets, of the ASCII-Origin field.

Origin: An OPTIONAL sequence of characters containing the ASCII serialization of an origin ([RFC6454], Section 6.2) that the sender asserts this connection is or could be authoritative for.

The ORIGIN frame does not define any flags. However, future updates to this specification MAY define flags. See Section 2.2.

2.2. Processing ORIGIN Frames

The ORIGIN frame is a non-critical extension to HTTP/2. Endpoints that do not support this frame can safely ignore it upon receipt.

When received by an implementing client, it is used to initialise and manipulate the Origin Set (see Section 2.3), thereby changing how the client establishes authority for origin servers (see Section 2.4).

The ORIGIN frame MUST be sent on stream 0; an ORIGIN frame on any other stream is invalid and MUST be ignored.

Likewise, the ORIGIN frame is only valid on connections with the "h2" protocol identifier, or when specifically nominated by the protocol's definition; it MUST be ignored when received on a connection with the "h2c" protocol identifier.

This specification does not define any flags for the ORIGIN frame, but future updates to this specification (through IETF consensus) might use them to change its semantics. The first four flags (0x1, 0x2, 0x4 and 0x8) are reserved for backwards-incompatible changes, and therefore when any of them are set, the ORIGIN frame containing them MUST be ignored by clients conforming to this specification, unless the flag's semantics are understood. The remaining flags are reserved for backwards-compatible changes, and do not affect processing by clients conformant to this specification.

The ORIGIN frame describes a property of the connection, and therefore is processed hop-by-hop. An intermediary MUST NOT forward ORIGIN frames. Clients configured to use a proxy MUST ignore any ORIGIN frames received from it.

Each ASCII-Origin field in the frame's payload MUST be parsed as an ASCII serialisation of an origin ([RFC6454], Section 6.2). If parsing fails, the field MUST be ignored.

Note that the ORIGIN frame does not support wildcard names (e.g., "*.example.com") in Origin-Entry. As a result, sending ORIGIN when a wildcard certificate is in use effectively disables any origins that are not explicitly listed in the ORIGIN frame(s) (when the client understands ORIGIN).

See Appendix A for an illustrative algorithm for processing ORIGIN frames.

2.3. The Origin Set

The set of origins (as per [RFC6454]) that a given connection might be used for is known in this specification as the Origin Set.

By default, the Origin Set for a connection is uninitialised. An uninitialized Origin Set means that clients apply the coalescing rules from Section 9.1.1 of [RFC7540].

When an ORIGIN frame is first received and successfully processed by a client, the connection's Origin Set is defined to contain an initial origin. The initial origin is composed from:

- Scheme: "https"
- Host: the value sent in Server Name Indication (SNI, [RFC6066], Section 3), converted to lower case; if SNI is not present, the remote address of the connection (i.e., the server's IP address)
- Port: the remote port of the connection (i.e., the server's port)

The contents of that ORIGIN frame (and subsequent ones) allows the server to incrementally add new origins to the Origin Set, as described in Section 2.2.

The Origin Set is also affected by the 421 (Misdirected Request) response status code, defined in [RFC7540], Section 9.1.2. Upon receipt of a response with this status code, implementing clients MUST create the ASCII serialisation of the corresponding request's origin (as per [RFC6454], Section 6.2) and remove it from the connection's Origin Set, if present.

Note: When sending an ORIGIN frame to a connection that is initialised as an Alternative Service [RFC7838], the initial origin set (Section 2.3) will contain an origin with the appropriate scheme and hostname (since Alternative Services specifies that the origin's hostname be sent in SNI). However, it is possible that the port will be different than that of the intended origin, since the initial origin set is calculated using the actual port in use, which can be different for the alternative service. In this case, the intended origin needs to be sent in the ORIGIN frame explicitly.

For example, a client making requests for "https://example.com" is directed to an alternative service at ("h2", "x.example.net", "8443"). If this alternative service sends an ORIGIN frame, the initial origin will be "https://example.com:8443". The client will not be able to use the alternative service to make requests

for "https://example.com" unless that origin is explicitly included in the ORIGIN frame.

2.4. Authority, Push and Coalescing with ORIGIN

Section 10.1 of [RFC7540] uses both DNS and the presented TLS certificate to establish the origin server(s) that a connection is authoritative for, just as HTTP/1.1 does in [RFC7230].

Furthermore, Section 9.1.1 of [RFC7540] explicitly allows a connection to be used for more than one origin server, if it is authoritative. This affects what responses can be considered authoritative, both for direct responses to requests and for server push (see [RFC7540], Section 8.2.2). Indirectly, it also affects what requests will be sent on a connection, since clients will generally only send requests on connections that they believe to be authoritative for the origin in question.

Once an Origin Set has been initialised for a connection, clients that implement this specification use it to help determine what the connection is authoritative for. Specifically, such clients MUST NOT consider a connection to be authoritative for an origin not present in the Origin Set, and SHOULD use the connection for all requests to origins in the Origin Set for which the connection is authoritative, unless there are operational reasons for opening a new connection.

Note that for a connection to be considered authoritative for a given origin, the server is still required to authenticate with certificate that passes suitable checks; see Section 9.1.1 of [RFC7540] for more information. This includes verifying that the host matches a "dNSName" value from the certificate "subjectAltName" field (using the rules defined in [RFC2818]; see also [RFC5280], Section 4.2.1.6).

Additionally, clients MAY avoid consulting DNS to establish the connection’s authority for new requests to origins in the Origin Set; however, those that do so face new risks, as explained in Section 4.

Because ORIGIN can change the set of origins a connection is used for over time, it is possible that a client might have more than one viable connection to an origin open at any time. When this occurs, clients SHOULD NOT emit new requests on any connection whose Origin Set is a proper subset of another connection’s Origin Set, and SHOULD close it once all outstanding requests are satisfied.

The Origin Set is unaffected by any alternative services [RFC7838] advertisements made by the server. Advertising an alternative service does not affect whether a server is authoritative.

3. IANA Considerations

This specification adds an entry to the "HTTP/2 Frame Type" registry.

- Frame Type: ORIGIN
- Code: 0xc
- Specification: [this document]

4. Security Considerations

Clients that blindly trust the ORIGIN frame's contents will be vulnerable to a large number of attacks. See Section 2.4 for mitigations.

Relaxing the requirement to consult DNS when determining authority for an origin means that an attacker who possesses a valid certificate no longer needs to be on-path to redirect traffic to them; instead of modifying DNS, they need only convince the user to visit another Web site in order to coalesce connections to the target onto their existing connection.

As a result, clients opting not to consult DNS ought to employ some alternative means to establish a high degree of confidence that the certificate is legitimate. For example, clients might skip consulting DNS only if they receive proof of inclusion in a Certificate Transparency log [RFC6962] or they have a recent OCSP response [RFC6960] (possibly using the "status_request" TLS extension [RFC6066]) showing that the certificate was not revoked.

The Origin Set's size is unbounded by this specification, and thus could be used by attackers to exhaust client resources. To mitigate this risk, clients can monitor their state commitment and close the connection if it is too high.

5. References

5.1. Normative References

[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>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[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, <https://www.rfc-editor.org/info/rfc5280>.

[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>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <https://www.rfc-editor.org/info/rfc6454>.

[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>.

5.2. Informative References

[RFC6960] Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 6960, DOI 10.17487/RFC6960, June 2013, <https://www.rfc-editor.org/info/rfc6960>.

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://www.rfc-editor.org/info/rfc6962>.

[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>.

[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>.

[RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

5.3. URIs

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

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

[3] https://github.com/httpwg/http-extensions/labels/origin-frame

Appendix A. Non-Normative Processing Algorithm

The following algorithm illustrates how a client could handle received ORIGIN frames:

1. If the client is configured to use a proxy for the connection, ignore the frame and stop processing.

2. If the connection is not identified with the "h2" protocol identifier or another protocol that has explicitly opted into this specification, ignore the frame and stop processing.

3. If the frame occurs upon any stream except stream 0, ignore the frame and stop processing.

4. If any of the flags 0x1, 0x2, 0x4 or 0x8 are set, ignore the frame and stop processing.

5. If no previous ORIGIN frame on the connection has reached this step, initialise the Origin Set as per Section 2.3.

6. For each "Origin-Entry" in the frame payload:

    1. Parse "ASCII-Origin" as an ASCII serialization of an origin ([RFC6454], Section 6.2) and let the result be "parsed_origin". If parsing fails, skip to the next "Origin-Entry".

    2. Add "parsed_origin" to the Origin Set.

Appendix B. Operational Considerations for Servers

The ORIGIN frame allows a server to indicate for which origins a given connection ought be used. The set of origins advertised using this mechanism is under control of the server; servers are not obligated to use it, or to advertise all origins which they might be able to answer a request for.

For example, it can be used to inform the client that the connection is to only be used for the SNI-based origin, by sending an empty

ORIGIN frame. Or, a larger number of origins can be indicated by including a payload.

Generally, this information is most useful to send before sending any part of a response that might initiate a new connection; for example, "Link" header fields [RFC8288] in a response HEADERS, or links in the response body.

Therefore, the ORIGIN frame ought be sent as soon as possible on a connection, ideally before any HEADERS or PUSH_PROMISE frames.

However, if it’s desirable to associate a large number of origins with a connection, doing so might introduce end-user perceived latency, due to their size. As a result, it might be necessary to select a "core" set of origins to send initially, expanding the set of origins the connection is used for with subsequent ORIGIN frames later (e.g., when the connection is idle).

That said, senders are encouraged to include as many origins as practical within a single ORIGIN frame; clients need to make decisions about creating connections on the fly, and if the origin set is split across many frames, their behaviour might be suboptimal.

Senders take note that, as per Section 4, Step 5 of [RFC6454], the values in an ORIGIN header need to be case-normalised before serialisation.

Finally, servers that host alternative services [RFC7838] will need to explicitly advertise their origins when sending ORIGIN, because the default contents of the Origin Set (as per Section 2.3) do not contain any Alternative Services’ origins, even if they have been used previously on the connection.

Authors’ Addresses

Mark Nottingham

Email: mnot@mnot.net
URI: https://www.mnot.net/

Erik Nygren
Akamai

Email: nygren@akamai.com

ï»¿

HTTP Working Group
Internet-Draft
Obsoletes: 5987 (if approved)
Intended status: Standards Track
Expires: September 3, 2017

J. Reschke
greenbytes

March 2, 2017

Indicating Character Encoding and Language for HTTP Header Field Parameters
draft-ietf-httpbis-rfc5987bis-05

Abstract

By default, header field values in Hypertext Transfer Protocol (HTTP) messages cannot easily carry characters outside the US-ASCII coded character set. RFC 2231 defines an encoding mechanism for use in parameters inside Multipurpose Internet Mail Extensions (MIME) header field values. This document specifies an encoding suitable for use in HTTP header fields that is compatible with a simplified profile of the encoding defined in RFC 2231.

This document obsoletes RFC 5987.

Editorial Note (To be removed by RFC Editor before publication)

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

Working Group information can be found at <http://httpwg.github.io/>; source code and issues list for this draft can be found at <https://github.com/httpwg/http-extensions>.

The changes in this draft are summarized in Appendix C.

Status of This Memo

material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 3, 2017.

Table of Contents

1. Introduction 4
2. Notational Conventions 4
3. Comparison to RFC 2231 and Definition of the Encoding 5
3.1. Parameter Continuations 5
3.2. Parameter Value Character Encoding and Language Information 5
3.2.1. Definition 6
3.2.2. Historical Notes 7
3.2.3. Examples 8
3.3. Language Specification in Encoded Words 8
4. Guidelines for Usage in HTTP Header Field Definitions 9
4.1. When to Use the Extension 9
4.2. Error Handling 10
5. Security Considerations 10
6. IANA Considerations 10
7. References 11
7.1. Normative References 11
7.2. Informative References 12
Appendix A. Changes from RFC 5987 13
Appendix B. Implementation Report 14
Appendix C. Change Log (to be removed by RFC Editor before publication) 14
C.1. Since RFC5987 15
C.2. Since draft-reschke-rfc5987bis-00 15
C.3. Since draft-reschke-rfc5987bis-01 15
C.4. Since draft-reschke-rfc5987bis-02 15
C.5. Since draft-reschke-rfc5987bis-03 15
C.6. Since draft-reschke-rfc5987bis-04 15
C.7. Since draft-reschke-rfc5987bis-05 15
C.8. Since draft-reschke-rfc5987bis-06 15
C.9. Since draft-ietf-httpbis-rfc5987bis-00 15
C.10. Since draft-ietf-httpbis-rfc5987bis-01 15
C.11. Since draft-ietf-httpbis-rfc5987bis-02 16
C.12. Since draft-ietf-httpbis-rfc5987bis-03 16
C.13. Since draft-ietf-httpbis-rfc5987bis-04 16
Appendix D. Acknowledgements 16

1. Introduction

Use of characters outside the US-ASCII coded character set ([RFC0020]) in HTTP header fields ([RFC7230]) is non-trivial:

- The HTTP specification discourages use of non-US-ASCII characters in field values, placing them into the "obs-text" ABNF production ([RFC7230], Section 3.2).

- Furthermore, it stays silent about default character encoding schemes for field values, so any use of non-US-ASCII characters would need to be specific to the field definition, or would require some other kind of out-of-band information.

- Finally, some APIs assume a default character encoding scheme in order to map from the octet sequences (obtained from the HTTP message) to character sequences: for instance, the XMLHttpRequest API ([XMLHttpRequest]) uses the Interface Definition Language type "ByteString", effectively resulting in the ISO-8859-1 character encoding scheme [ISO-8859-1] being used.

On the other hand, RFC 2231 defines an encoding mechanism for parameters inside MIME header fields ([RFC2231]), which, as opposed to HTTP messages, do need to be sent over non-binary transports. This document specifies an encoding suitable for use in HTTP header fields that is compatible with a simplified profile of the encoding defined in RFC 2231. It can be applied to any HTTP header field that uses the common "parameter" ("name=value") syntax.

This document obsoletes [RFC5987] and moves it to "historic" status; the changes are summarized in Appendix A.

> Note: in the remainder of this document, RFC 2231 is only referenced for the purpose of explaining the choice of features that were adopted; they are therefore purely informative.
>
> Note: this encoding does not apply to message payloads transmitted over HTTP, such as when using the media type "multipart/form-data" ([RFC7578]).

2. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This specification uses the ABNF (Augmented Backus-Naur Form) notation defined in [RFC5234]. The following core rules are included

by reference, as defined in [RFC5234], Appendix B.1: ALPHA (letters), DIGIT (decimal 0-9), HEXDIG (hexadecimal 0-9/A-F/a-f), and LWSP (linear whitespace).

This specification uses terminology defined in [RFC6365], namely: "character encoding scheme" (below abbreviated to "character encoding"), "charset" and "coded character set".

Note that this differs from RFC 2231, which uses the term "character set" for "character encoding scheme".

3. Comparison to RFC 2231 and Definition of the Encoding

RFC 2231 defines several extensions to MIME. The sections below discuss if and how they apply to HTTP header fields.

In short:

- Parameter Continuations aren’t needed (Section 3.1),
- Character Encoding and Language Information are useful, therefore a simple subset is specified (Section 3.2), and
- Language Specifications in Encoded Words aren’t needed (Section 3.3).

3.1. Parameter Continuations

Section 3 of [RFC2231] defines a mechanism that deals with the length limitations that apply to MIME headers. These limitations do not apply to HTTP ([RFC7231], Appendix A.6).

Thus, parameter continuations are not part of the encoding defined by this specification.

3.2. Parameter Value Character Encoding and Language Information

Section 4 of [RFC2231] specifies how to embed language information into parameter values, and also how to encode non-ASCII characters, dealing with restrictions both in MIME and HTTP header field parameters.

However, RFC 2231 does not specify a mandatory-to-implement character encoding, making it hard for senders to decide which encoding to use. Thus, recipients implementing this specification MUST support the "UTF-8" character encoding [RFC3629].

Furthermore, RFC 2231 allows the character encoding information to be

left out. The encoding defined by this specification does not allow that.

3.2.1. Definition

The presence of extended parameter values usually is indicated by a parameter name ending in an asterisk character. Note however that this is just a convention, and that it needs to be explicitly specified in the definition of the header field using this extension (see Section 4).

The ABNF for extended parameter values is specified below:

```
ext-value     = charset  "'" [ language ] "'" value-chars
              ; like RFC 2231's <extended-initial-value>
              ; (see [RFC2231], Section 7)

charset       = "UTF-8" / mime-charset

mime-charset  = 1*mime-charsetc
mime-charsetc = ALPHA / DIGIT
              / "!" / "#" / "$" / "%" / "&"
              / "+" / "-" / "^" / "_" / "'"
              / "{" / "}" / "~"
              ; as <mime-charset> in Section 2.3 of [RFC2978]
              ; except that the single quote is not included
              ; SHOULD be registered in the IANA charset registry

language      = <Language-Tag, see [RFC5646], Section 2.1>

value-chars   = *( pct-encoded / attr-char )

pct-encoded   = "%" HEXDIG HEXDIG
              ; see [RFC3986], Section 2.1

attr-char     = ALPHA / DIGIT
              / "!" / "#" / "$" / "&" / "+" / "-" / "."
              / "^" / "_" / "'" / "|" / "~"
              ; token except ( "*" / "'" / "%" )
```

The value part of an extended parameter (ext-value) is a token that consists of three parts:

1. the REQUIRED character encoding name (charset),

2. the OPTIONAL language information (language), and

3. a character sequence representing the actual value (value-chars), separated by single quote characters.

Note that both character encoding names and language tags are restricted to the US-ASCII coded character set, and are matched case-insensitively (see [RFC2978], Section 2.3 and [RFC5646], Section 2.1.1).

Inside the value part, characters not contained in attr-char are encoded into an octet sequence using the specified character encoding. That octet sequence is then percent-encoded as specified in Section 2.1 of [RFC3986].

Producers MUST use the "UTF-8" ([RFC3629]) character encoding. Extension character encodings (mime-charset) are reserved for future use.

Note: recipients should be prepared to handle encoding errors, such as malformed or incomplete percent escape sequences, or non-decodable octet sequences, in a robust manner. This specification does not mandate any specific behavior, for instance, the following strategies are all acceptable:

* ignoring the parameter,

* stripping a non-decodable octet sequence,

* substituting a non-decodable octet sequence by a replacement character, such as the Unicode character U+FFFD (Replacement Character).

3.2.2. Historical Notes

The RFC 7230 token production ([RFC7230], Section 3.2.6) differs from the production used in RFC 2231 (imported from Section 5.1 of [RFC2045]) in that curly braces ("{" and "}") are excluded. Thus, these two characters are excluded from the attr-char production as well.

The <mime-charset> ABNF defined here differs from the one in Section 2.3 of [RFC2978] in that it does not allow the single quote character (see also RFC Errata ID 1912 [Err1912]). In practice, no character encoding names using that character have been registered at the time of this writing.

For backwards compatibility with RFC 2231, the encoding defined by this specification deviates from common parameter syntax in that the quoted-string notation is not allowed. Implementations using generic

parser components might not be able to detect the use of quoted-string notation and thus might accept that format, although invalid, as well.

[RFC5987] did require support for ISO-8859-1 ([ISO-8859-1]), too; for compatibility with legacy code, recipients are encouraged to support this encoding as well.

3.2.3. Examples

Non-extended notation, using "token":

```
foo: bar; title=Economy
```

Non-extended notation, using "quoted-string":

```
foo: bar; title="US-$ rates"
```

Extended notation, using the Unicode character U+00A3 ("Â£", POUND SIGN):

```
foo: bar; title*=utf-8'en'%C2%A3%20rates
```

Note: the Unicode pound sign character U+00A3 was encoded into the octet sequence C2 A3 using the UTF-8 character encoding, then percent-encoded. Also, note that the space character was encoded as %20, as it is not contained in attr-char.

Extended notation, using the Unicode characters U+00A3 ("Â£", POUND SIGN) and U+20AC ("â\202¬", EURO SIGN):

```
foo: bar; title*=UTF-8''%c2%a3%20and%20%e2%82%ac%20rates
```

Note: the Unicode pound sign character U+00A3 was encoded into the octet sequence C2 A3 using the UTF-8 character encoding, then percent-encoded. Likewise, the Unicode euro sign character U+20AC was encoded into the octet sequence E2 82 AC, then percent-encoded. Also note that HEXDIG allows both lowercase and uppercase characters, so recipients must understand both, and that the language information is optional, while the character encoding is not.

3.3. Language Specification in Encoded Words

Section 5 of [RFC2231] extends the encoding defined in [RFC2047] to also support language specification in encoded words. RFC 2616, the now-obsolete HTTP/1.1 specification, did refer to RFC 2047 ([RFC2616], Section 2.2). However, it wasn't clear to which header field it applied. Consequently, the current revision of the HTTP/1.1

specification has deprecated use of the encoding forms defined in RFC 2047 (see Section 3.2.4 of [RFC7230]).

Thus, this specification does not include this feature.

4. Guidelines for Usage in HTTP Header Field Definitions

Specifications of HTTP header fields that use the extensions defined in Section 3.2 ought to clearly state that. A simple way to achieve this is to normatively reference this specification, and to include the ext-value production into the ABNF for specific header field parameters.

For instance:

```
foo         = token ";" LWSP title-param
title-param = "title" LWSP "=" LWSP value
            / "title*" LWSP "=" LWSP ext-value
ext-value   = <see draft-ietf-httpbis-rfc5987bis, Section 3.2>
```

[[pub: Upon publication as RFC, the string "draft-ietf-httpbis-rfc5987bis" needs to be replaced with the RFC name, and this comment needs to be removed.]]

> Note: The Parameter Value Continuation feature defined in Section 3 of [RFC2231] makes it impossible to have multiple instances of extended parameters with identical names, as the processing of continuations would become ambiguous. Thus, specifications using this extension are advised to disallow this case for compatibility with RFC 2231.
>
> Note: This specification does not automatically assign a new interpretation to parameter names ending in an asterisk. As pointed out above, it’s up to the specification for the non-extended parameter to "opt in" to the syntax defined here. That being said, some existing implementations are known to automatically switch to the use of this notation when a parameter name ends with an asterisk, thus using parameter names ending in an asterisk for something else is likely to cause interoperability problems.

4.1. When to Use the Extension

Section 4.2 of [RFC2277] requires that protocol elements containing human-readable text are able to carry language information. Thus, the ext-value production ought to be always used when the parameter value is of textual nature and its language is known.

Furthermore, the extension ought to also be used whenever the parameter value needs to carry characters not present in the US-ASCII ([RFC0020]) coded character set (note that it would be unacceptable to define a new parameter that would be restricted to a subset of the Unicode character set).

4.2. Error Handling

Header field specifications need to define whether multiple instances of parameters with identical names are allowed, and how they should be processed. This specification suggests that a parameter using the extended syntax takes precedence. This would allow producers to use both formats without breaking recipients that do not understand the extended syntax yet.

Example:

```
  foo: bar; title="EURO exchange rates";
            title*=utf-8''%e2%82%ac%20exchange%20rates
```

In this case, the sender provides an ASCII version of the title for legacy recipients, but also includes an internationalized version for recipients understanding this specification -- the latter obviously ought to prefer the new syntax over the old one.

5. Security Considerations

The format described in this document makes it possible to transport non-ASCII characters, and thus enables character "spoofing" scenarios, in which a displayed value appears to be something other than it is.

Furthermore, there are known attack scenarios relating to decoding UTF-8.

See Section 10 of [RFC3629] for more information on both topics.

In addition, the extension specified in this document makes it possible to transport multiple language variants for a single parameter, and such use might allow spoofing attacks, where different language versions of the same parameter are not equivalent. Whether this attack is useful as an attack depends on the parameter specified.

6. IANA Considerations

There are no IANA Considerations related to this specification.

7. References

7.1. Normative References

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <http://www.rfc-editor.org/info/rfc20>.

[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>.

[RFC2978] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, DOI 10.17487/RFC2978, October 2000, <http://www.rfc-editor.org/info/rfc2978>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <http://www.rfc-editor.org/info/rfc5646>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231,

June 2014,
<http://www.rfc-editor.org/info/rfc7231>.

7.2. Informative References

[Err1912] RFC Errata, "Errata ID 1912, RFC 2978", October 2009, <http://www.rfc-editor.org>.

[ISO-8859-1] International Organization for Standardization, "Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1", ISO/IEC 8859-1:1998, 1998.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, DOI 10.17487/RFC2047, November 1996, <http://www.rfc-editor.org/info/rfc2047>.

[RFC2231] Freed, N. and K. Moore, "MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations", RFC 2231, DOI 10.17487/RFC2231, November 1997, <http://www.rfc-editor.org/info/rfc2231>.

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, DOI 10.17487/RFC2277, January 1998, <http://www.rfc-editor.org/info/rfc2277>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", RFC 5987, DOI 10.17487/RFC5987, August 2010, <http://www.rfc-editor.org/info/rfc5987>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6266] Reschke, J., "Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP)", RFC 6266, DOI 10.17487/RFC6266, June 2011, <http://www.rfc-editor.org/info/rfc6266>.

[RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <http://www.rfc-editor.org/info/rfc6365>.

[RFC7578] Masinter, L., "Returning Values from Forms: multipart/form-data", RFC 7578, DOI 10.17487/RFC7578, July 2015, <http://www.rfc-editor.org/info/rfc7578>.

[RFC7616] Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", RFC 7616, DOI 10.17487/RFC7616, September 2015, <http://www.rfc-editor.org/info/rfc7616>.

[RFC8053] Oiwa, Y., Watanabe, H., Takagi, H., Maeda, K., Hayashi, T., and Y. Ioku, "HTTP Authentication Extensions for Interactive Clients", RFC 8053, DOI 10.17487/RFC8053, January 2017, <http://www.rfc-editor.org/info/rfc8053>.

[XMLHttpRequest] WhatWG, "XMLHttpRequest", <https://xhr.spec.whatwg.org/>.

Appendix A. Changes from RFC 5987

This section summarizes the changes compared to [RFC5987]:

- The document title was changed to "Indicating Character Encoding and Language for HTTP Header Field Parameters".
- The introduction was rewritten to better explain the issues around non-ASCII characters in field values.
- The requirement to support the "ISO-8859-1" encoding was removed.
- The document does not attempt to re-define a generic "parameter" ABNF anymore (it turned out that there really isn’t a generic definition of parameters in HTTP; for instance, there are subtle

differences with respect to whitespace handling).

- A note about defects in error handling in current implementations was removed, as it wasn't accurate anymore.

Appendix B. Implementation Report

The encoding defined in this document currently is used in four different HTTP header fields:

- "Authentication-Control", defined in [RFC8053],
- "Authorization" (as used in HTTP Digest Authentication, defined in [RFC7616]),
- "Content-Disposition", defined in [RFC6266], and
- "Link", defined in [RFC5988].

As the encoding is a profile/clarification of the one defined in [RFC2231] in 1997, many user agents already supported it for use in "Content-Disposition" when [RFC5987] got published.

Since the publication of [RFC5987], three more popular desktop user agents have added support for this encoding; see <http://purl.org/NET/http/content-disposition-tests#encoding-2231-char> for details. At this time, the current versions of all major desktop user agents support it.

Note that the implementation in Internet Explorer 9 does not support the ISO-8859-1 character encoding; this document revision acknowledges that UTF-8 is sufficient for expressing all code points, and removes the requirement to support ISO-8859-1.

The "Link" header field, on the other hand, was more recently specified in [RFC5988]. At the time of this writing, no User Agent except Firefox supported the "title*" parameter (starting with release 15).

Section 3.4 of [RFC7616] defines the "username*" parameter for use in HTTP Digest Authentication. At the time of writing, no User Agent implemented this extension.

Appendix C. Change Log (to be removed by RFC Editor before publication)

C.1. Since RFC5987

Only editorial changes for the purpose of starting the revision process (obs5987).

C.2. Since draft-reschke-rfc5987bis-00

Resolved issues "iso-8859-1" and "title" (title simplified). Added and resolved issue "historic5987".

C.3. Since draft-reschke-rfc5987bis-01

Added issues "httpbis", "parmsyntax", "terminology" and "valuesyntax". Closed issue "impls".

C.4. Since draft-reschke-rfc5987bis-02

Resolved issue "terminology".

C.5. Since draft-reschke-rfc5987bis-03

In Section 3.2, pull historical notes into a separate subsection. Resolved issues "valuesyntax" and "parmsyntax".

C.6. Since draft-reschke-rfc5987bis-04

Update status of Firefox support in HTTP Link Header field.

C.7. Since draft-reschke-rfc5987bis-05

Update status of Firefox support in HTTP Link Header field.

C.8. Since draft-reschke-rfc5987bis-06

Update status with respect to Safari 6.

Started work on update with respect to RFC 723x.

C.9. Since draft-ietf-httpbis-rfc5987bis-00

Editorial changes; introducing non-ASCII characters into author’s address, acknowledgements, and examples.

C.10. Since draft-ietf-httpbis-rfc5987bis-01

Removed mention of RFC 2616 from Abstract and Introduction.

Reference RFC 20 for US-ASCII.

Do not attempt to define a generic parameter ABNF; just concentrate on the parameter value syntax.

C.11. Since draft-ietf-httpbis-rfc5987bis-02

RFC 2388 -> RFC 7578.

Expand on the motivation (see <https://github.com/httpwg/http-extensions/issues/213>).

Mention RFC 7616 in implementation report.

C.12. Since draft-ietf-httpbis-rfc5987bis-03

Fixed one editorial issue. Updated XHR reference.

Fixed <https://github.com/httpwg/http-extensions/issues/266>: use of now undefined term "parmname".

Include WG into Acknowledgements for this revision.

Mention RFC 5987 in the abstract (<https://github.com/httpwg/http-extensions/issues/271>).

C.13. Since draft-ietf-httpbis-rfc5987bis-04

Mention RFC8053 in Implementation Report.

Appendix D. Acknowledgements

Thanks to Martin DÃ¼rst and Frank Ellermann for help figuring out ABNF details, to Graham Klyne and Alexey Melnikov for general review, to Chris Newman for pointing out an RFC 2231 incompatibility, and to Benjamin Carlyle, Roar Lauritzsen, Eric Lawrence, and James Manger for implementer’s feedback.

Furthermore thanks to the members of the IETF HTTP Working Group for the feedback specific to this update of RFC 5987.

Author's Address

Julian F. Reschke
greenbytes GmbH
Hafenweg 16
MÃ¼nster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: http://greenbytes.de/tech/webdav/

Network Working Group
Internet-Draft
Intended status: Informational
Expires: January 9, 2017

G. Montenegro
Microsoft
S. Cespedes
Universidad de Chile
S. Loreto
Ericsson
R. Simpson
General Electric

July 8, 2016

H2oT: HTTP/2 for the Internet of Things
draft-montenegro-httpbis-h2ot-00

Abstract

This document makes the case for HTTP/2 for IoT.

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 January 9, 2017.

Copyright Notice

Table of Contents

1. Introduction 2
2. Application Transport Alternatives and their Strengths . . . 4
2.1. HTTP/1.1 5
2.2. MQTT 5
2.3. CoAP 6
2.4. Protocols Comparison 8
3. Importance of Protocol Reuse 8
4. HTTP/2 in IoT 10
5. Profile of HTTP/2 for IoT 11
6. Negotiation of HTTP/2 for IoT 12
7. Gateway and Proxying Issues 12
8. Implementation Considerations 13
9. Experimentation and Performance 13
9.1. GET Example 14
9.1.1. HTTP/1.1 14
9.1.2. HTTP/2 15
9.1.3. Comparison 16
10. HTTP/2 over UDP - QUIC 16
11. IANA Considerations 17
12. Security Considerations 17
13. Acknowledgements 18
14. References 18
14.1. Normative References 18
14.2. Informative References 19
Authors' Addresses 21

1. Introduction

When the IETF started work on the Internet-of-Things ("IoT") with the 6lowpan WG, it was clear that in addition to the lower-layer adaptation work for IPv6, much work elsewhere in the stack was necessary. (In this document, the "things" in "IoT" are nodes that are constrained in some manner--e.g., cpu, memory, power--such that direct use of unmodified mainstream protocols is challenging.) Once the IPv6 adaptation was understood, the next question was what protocols to use above IP(v6) for different functions and at different layers to have a complete stack. That question may not have a single answer that is always best for all scenarios and use cases. There are many such use cases, in accordance with the fact that IoT means too many things.

Accordingly, the IoT landscape includes a proliferation of options for any particular functionality (transport, encoding, security

suites, authentication/authorization, etc). Different vendors and standards organizations (or fora) offer IoT solutions by grouping these different components into separate stacks. Even if the components have the same name or originate in the same original standard (or even in the same code base), each organization adapts it ever so slightly to their own goals, often rendering the resultant components non-interoperable. Many of these components are being created expressly for IoT (within the IETF and elsewhere) under the assumption that the mainstream options could not possibly be usable for IoT scenarios. This results in multiple disparate networking and software stacks. Given the incipient state of IoT, for the foreseeable future multiple competing stacks will continue to exist at least in gateways and cloud elements. The additional complexity to IoT amplifies the attack surface. Nevertheless, properly configured and implemented, mainstream options may not just be workable, but may even be the best option at least in some scenarios.

The appearance of one-off stacks (as opposed to a properly configured and adapted mainstream stack) is reminiscent of WAP 1.x, a complete vertical stack offered for phones as they were starting to access the Internet (albeit from within a walled garden) in the late 90's. At that time the IETF and the W3C started efforts to develop the mainstream alternatives. As a result, today no phone uses WAP. Instead, phone stacks are mainstream TCP/IP protocols (properly configured and adapted, of course). In contrast, today in IoT we see not just one non-mainstream stack, but several (as if we had not just WAP, but WAP1, WAP2, WAP3, etc.). And we may have to live with them for some time, but it is essential to ponder what the mainstream stack might look like if we are to eventually reap the benefits of a true Internet of Things instead of a not-quite-but-kinda-close-to-Internet-non-interoperable-hodge-podge-of-Things.

HTTP/2 [RFC7540][RFC7541] is now widely available as a transport option. Moreover, the ongoing effort to layer HTTP/2 over UDP (i.e., over QUIC) adds a useful capability for IoT scenarios. We show the current suitability of HTTP/2 for IoT scenarios and examine possible improvements.

Let's look at some application communication patterns to establish some common language (see also section 2 of [RFC7452] for a related discussion):

node to node: A constrained node engages in direct communication with another constrained node.

node to gateway: A constrained node and a gateway node engage in direct communication. A gateway node is directly on both a constrained network (e.g., a lowpan) and on a non-constrained

network (a normal network using mainstream stack implementations, typically connected to the Internet).

gateway to cloud: A gateway node (see above) engages in communication with unconstrained networks, typically a cloud service on the Internet.

node to cloud: A node on a constrained network engages in direct communcation with unconstrained networks, typically a cloud service on the Internet.

In the above, a "node" may, in fact, be multiple nodes when engaging in group communication. Group communications (e.g., via multicast) are commonly used for discovery or routing (see also Section 2.3).

We can further categorize the above communication patterns into two basic types of networking exchanges:

Constrained network scenario: A constrained network scenario includes node to node and node to gateway exchanges. Group communications are another typical aspect of these constrained networks.

Internet scenario: An Internet scenario includes gateway to cloud and node to cloud exchanges.

This document makes the case for HTTP/2 as the most general protocol of choice for Internet of Things applications. HTTP/2 is most at home in Internet scenarios and is also suitable for at least some Constrained network scenarios.

2. Application Transport Alternatives and their Strengths

A recent survey by the Eclipse IoT working group queried IoT developers about the protocols and technologies they are using and planning to use [Eclipse_survey]. Some of the currently used application transport protocols (above the link layer) for IoT applications are as follows:

- HTTP/1.1 (61% of developers)
- MQTT (52% of developers)
- CoAP (21% of developers)
- HTTP/2 (19% of developers)
- Others: In-house, AMQP and XMPP (43% of developers)

It is interesting to note that in the same survey done in 2015, HTTP/2 was not even present, whereas it is now at 19% (the other protocols are mostly unchanged). No doubt it is being used in scenarios where there are no major constraints (precisely where HTTP/1.x is also being used). Optimizing it for IoT can further promote its use. The sections below provide some more details on top-of-the-list protocols other than HTTP/2.

2.1. HTTP/1.1

HTTP/1.1 is a text-based protocol, and is widely successful as it is the basis not just for the web, but for much non-web traffic in the internet today. Most (but not all) of the instances of HTTP today implement version 1.1 as specified in RFC2616 [RFC2616]. Since its publication back in 1999 it has evolved organically, producing countless variations and exceptions to its rules. Modern browser and server implementations have very complex and convoluted code to deal with parsing and handling the many nuances of the protocol. Because of all this confusion, the HTTPbis working group set out to clarify the existing specifications, and after a multi-year effort to clarify its many sources of confusion, it has published a cleaner specification in RFCs 7230-7235 [RFC7230] [RFC7231] [RFC7232] [RFC7233] [RFC7234] [RFC7235]. In spite of this, the protocol still has a plethora of legacy issues and remains too verbose.

HTTP/1.1 is very clearly a mismatch for the constrained devices and networks that characterize IoT. Despite its shortcomings, it is the most popular protocol for IoT applications (61% per the aforementioned survey, although the survey does not clarify if this is for Internet or constrained network scenarios). Why would such an ill-suited protocol be clearly the most popular for IoT applications? It is by far the most commonly known protocol. It has many implementations (many in open source), with massive support in all platforms, tools and APIs. It is easy to find know-how and support. In short, it has the power and convenience that comes with being a mainstream protocol.

Another major advantage is that it is the protocol that has the best chance of traversing firewalls and middle boxes in the internet due to its use of port 80 when in the clear, and, especially, its use of port 443 when over TLS. This is a primary concern in Internet scenarios.

2.2. MQTT

MQ Telemetry Transport (MQTT) is a publish/subscribe messaging protocol that runs on top of TCP. It was created by IBM. Version 3.1.1 is available as an OASIS standard [mqtt_oasis] and as an ISO

publication [mqtt_iso]. It is popular in the Internet scenario (node to cloud, gateway to cloud) and it aims to connect embedded devices and networks with applications and middleware. It is a compact, binary protocol, and is very popular in certain application domains. It has been known as a protocol suitable to be used in resource constrained devices and unreliable networks.

It is the second most popular protocol in the survey (behind HTTP/1.1) with 52% of developers using it. In the internet scenario, however, TLS is probably required. This additional TLS overhead renders all protocols slightly larger, so, e.g., MQTT loses some relative size advantage.

The MQTT protocol requires an underlying transport that provides an ordered, lossless, stream of bytes from the Client to Server and Server to Client. It cannot be used over UDP. There is an alternative (and not standardized) variant called MQTT-SN (previously called MQTT-S) which can use UDP, Zigbee or other datagram transports, but this is a substantially different protocol which has been tailored to meet the needs of small, battery-powered sensors connected by wireless sensor networks (WSNs), and relies upon a MQTT-SN Gateway or forwarder for external communications.

MQTT is closely tied to PUBLISH/SUBSCRIBE operations and this is the only mode of message transfer. This means that MQTT cannot be used for "node to node" communications because a server is required (the server forwards messages between publishers and subscribers, manages subscriptions, and performs user authorization functions.) The exclusive use of publish/subscribe operations can complicate some IoT operations, such as request-response traffic, and transferring large payloads (e.g. firmware updates). It is sometimes desirable to use a different protocol (like HTTP) for transferring large payloads, even though MQTT supports a maximum per-message payload size of 256 MiB. The OASIS MQTT-TC is considering proposals involving changes in handling request-response traffic and large message transfers.

MQTT is deployed over TCP (port 8833 when over TLS, port 1883 without TLS). Even when using TLS, it has the well-known firewall traversal issues common to any protocol not over port 443.

2.3. CoAP

CoAP is a compact, binary, UDP-based protocol based on RESTful principles and closely patterned after HTTP. It has been designed to be used in constrained devices and constrained networks. The protocol specification has been published [RFC7252], although additional functionalities such as congestion control, block-wise

transfer, TCP and TLS transfer and HTTP mapping are still being specified.

The protocol meets IoT requirements through the modification of some HTTP functionalities to achieve low-power consumption and operation over lossy links. To avoid undesirable packet fragmentation the CoAP specification provides an upper bound to the message size, dictating that a CoAP message, appropriately encapsulated, SHOULD fit within a single IP datagram:

> If the Path MTU is not known for a destination, an IP MTU of 1280 bytes SHOULD be assumed; if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size.

CoAP interaction with HTTP must traverse a proxy, with the concomitant issues of breaking end-to-end security (Section 7), but at least the common REST architecture makes it easier.

CoAP works on port 5683 and offers optional reliable delivery (thru a retransmission mechanism), support for unicast and multicast, and asynchronous message exchange. Multicast (see Section 1 is typically used by IoT SDO's for routing and discovery. A common use of multicast within CoAP is for discovery, something addressed in mainstream (and even some IoT) scenarios via mDNS [RFC6762] and DNS-SD [RFC6763]. More general uses of multicast within CoAP (and, in general, at the application transport, e.g., to address group communication for IoT), introduces complexity for security, IPv6 scoping, wireless reliability, etc.

A typical CoAP message can be between 10 and 20 bytes.

It is the third most popular protocol in the survey with a 21% preference. Nevertheless, since CoAP is UDP-based, in the Internet scenario it also suffers from firewall traversal issues, verbosity (as compared to TCP) to maintain state in NAT boxes and lack of integration with existing enterprise infrastructures. There is ongoing work to specify the use of CoAP over TCP as well as CoAP over TLS, in an attempt to overcome issues with middleboxes and improve its applicability to Internet scenarios.

As noted above, there is much ongoing work on CoAP, and much of it seeks to define a transport on top of UDP. This is a very complex task not to be underestimated. QUIC is also embarking on this task, but it appears to be benefitting from many more resources within the networking community at large.

2.4. Protocols Comparison

The aformentioned protocols have been compared in both experimental and emulated environments [IEEE_survey]. Previous reports show that performance is highly dependent on the network conditions: in good link conditions with low packet loss, MQTT delivers packets with lower delay than CoAP, but CoAP outperforms when high packet losses are present; in terms of packet sizes, if packet loss is under 25% and messages are of a small size, CoAP demonstrates a better link usage than MQTT. However, other experiments report a better performance of MQTT in high traffic/high packet loss scenarios [IoT_analysis]. CoAP has also been compared to HTTP/1.1. In terms of power consumption and response time, naturally CoAP behaves better than HTTP/1.1 thanks to the reduced packet sizes.

Coexistence among the protocols has also been tested with varied network configurations. For the most part, interaction of CoAP with HTTP has been studied [Web_things], demonstrating successful exchange when there is a CoAP server running on a constrained node and the HTTP client is requesting resources from it, or when there is a CoAP client requesting resources from an HTTP server. In both cases a proxy is necessary to enable translation between the protocols. Another network configuration with a CoAP client - CoAP proxy - CoAP server has been compared to the CoAP client - CoAP/HTTP proxy - HTTP server configuration, in which case the response times of the only-CoAP configuration resulted to be lower even when the number of concurrent requests increases [CoAP_integration].

To date, no reports have been found comparing MQTT or CoAP to HTTP/2.

3. Importance of Protocol Reuse

These protocols often do not exist in a vacuum. Typically, they are mandated as part of a given stack specified by any of several IoT consortia (e.g., OCF, AllSeen Alliance/AllJoyn, Thread Group). We know that these multiple IoT protocols (and stacks) provide very useful sources of information for prying eyes (See "US intelligence chief says we might use the IoT to spy on you" at http://www.wired.com/2012/03/petraeus-tv-remote/). Security and privacy issues are exacerbated because:

- IoT is the worst of all security worlds: (1) constraints push devices into taking shortcuts and (2) there is less physical security with such devices (after installation they are typically reachable by unfriendly hands).

- Each of these protocols is an island with its own security measures (or lack thereof) and very limited review.

The previous two points can be summarized as follows:

A security and privacy environment even more challenging than usual:

> This is receiving much attention from the research and standardization communities. It is the sort of challenge that stimulates researchers into high gear. It is a daunting problem for sure, but at least it is on the radar of folks and consortia working on IoT. Nevertheless, many issues will arise because of this (e.g., discovery of serious flaws in IoT devices like locks is a common occurrence).

Many different protocol stacks at play: This is a much more worrisome issue if one considers that a vast majority of issues arising with security have less to do with cryptography (the first point above) and more to do with software engineering, and silly bugs. Each stack added creates more attack surface. At the same time, each one of these stacks gather less attention and scrutiny than software used for mainstream scenarios (such as the web). We have seen no shortage of issues on OpenSSL and similar heavily-used software. We can expect much worse from stacks that are not nearly as well exercised nor examined. And if we have not one, but several of these stacks untested by millions of eyeballs, we are inviting disaster.

A recent Harvard report on the state of surveillance and erosion of privacy [Going_Dark] concludes among its findings that the projected substantial growth of IoT will drastically change surveillance (surveillance is not merely limited to government agencies of course), and that the fragmentation of ecosystems hinders the deployment of countermeasures (e.g., end-to-end encryption) as that requires more coordination and standardization than currently available. This not only gives rise to rogue surveillance sites such as Shodan (https://www.shodan.io/), but also represents a great opportunity for government agencies' surveillance needs [Clapper].

Furthermore, multiple stacks defeat one of the main benefits of the "I" in IoT: interoperability. Also, reusing mainstream protocols affords the benefits of using better-known technology, with easier access to reference implementations (including open source), people with the required skills and experience, training, etc. These are basically the same arguments that were used originally to justify the use of IP-based networking over custom-built stacks. The message was heard loud and clear but for the most part it was applied to only a limited set of components (e.g., IP, UDP, DTLS). Other components are still being custom built (albeit, on top of IP).

4. HTTP/2 in IoT

As noted above, for the foreseeable future the IoT landscape requires several stacks. Thinking about a canonical stack based on mainstream protocols is not an exercise in the delusion that one single stack will be enough. Rather, it is an attempt to define an option that can serve IoT better into the future, and one that can be recommended whenever there is a choice (often there isn't one).

The goals in pursuing a canonical stack are the following:

- Maximize Standards-based Elements across technologies and IoT stacks
- Reduce IoT protocol idiosyncrasies and specificity
- Reduce the number of "translators" needed in an IoT hub

To arrive at a canonical stack the mainstream standards-based stack must be properly profiled and optimized. This requires optimizing aspects such as:

- Authentication and Authorization framework by adapting OAUTH instead of inventing a new system [I-D.ietf-ace-oauth-authz].
- Device Management and Object Model/Descriptions (currently being defined).
- Discovery via mDNS [RFC6762] and DNS-SD [RFC6763] perhaps augmented with IoT considerations, e.g., [I-D.aggarwal-dnssd-optimize-query]. Another option to investigate is that of HTTP/2 over multicast. Whereas there have been some forays into HTTP over multicast, it is not nearly as well deployed, implemented and understood as mDNS.
- Application Transport based on HTTP/2.

This document deals only with the application layer transport based on HTTP/2.

HTTP/2 is a good match for IoT for several reasons:

- Binary and Compact (9 byte header)
- Header Compression [RFC7541]
- Traversal past firewalls/middle boxes via TLS over port 443

- Support of RESTful model in major development frameworks
- Know-how widely available

5. Profile of HTTP/2 for IoT

HTTP/2 has many negotiable settings that can improve its performance for IoT applications by reducing bandwidth, codespace, and RAM requirements. Specifically, the following settings and values have been found to be useful in IoT applications:

- SETTINGS_HEADER_TABLE_SIZE: this setting allows hosts to limit the size of the header compression table used for decoding, reducing required RAM, but potentially increasing bandwidth requirements. Initial value per HTTP/2 is 4096. IoT scenarios might benefit from changing this to a smaller value (e.g., 512), however, to avoid increased bandwidth usage, IoT scenarios should judiciously use HTTP headers and the dynamic header table [RFC7541].

- SETTINGS_ENABLE_PUSH: This setting allows clients to enable or disable server push. This functionality may not be required in some IoT applications. The initial value per HTTP/2 is 1.

- SETTINGS_MAX_CONCURRENT_STREAMS: this setting allows a sender to limit the number of simultaneous streams that a receiver can create for a connection. HTTP/2 recommends this value be no smaller than 100. IoT scenarios may wish to limit this to a much smaller number, such as 2 or 3.

- SETTINGS_INITIAL_WINDOW_SIZE: this setting allows hosts to limit the flow control window, potentially reducing buffer requirements at the expense of potentially underutilized bandwidth-delay products. Per HTTP/2 the initial value is 2^16-1 (65,535) octets. IoT scenarios may wish to limit this to smaller values in accordance with the node’s constraints (e.g., a few kilo-octets).

- SETTINGS_MAX_FRAME_SIZE: this setting allows hosts to specify the largest frame size they are willing to receive. Per HTTP/2 the initial value is 2^14 (16,384) octets. Somewhat counterintuitively, IoT hosts may wish to leave this value large and rely on flow control to avoid unnecessary framing overhead.

- SETTINGS_MAX_HEADER_LIST_SIZE: this setting allows hosts to limit the maximum size of the header list they are willing to receive. Per HTTP/2 the initial value of this setting is unlimited. IoT scenarios may wish to limit this to smaller values in accordance with the node’s constraints (e.g., a few kilo-octets).

6. Negotiation of HTTP/2 for IoT

For Constrained and Internet scenarios, it is assumed that HTTP/2 runs over TLS. Accordingly, the ALPN negotiation in section 3.3 of [RFC7540] applies. As seen above, an IoT scenario may wish to depart from the default SETTINGS. To do so, the usual SETTINGS negotiation applies. In this case, the initial SETTINGS negotiation setup is based on the first message exchange initiated by the client. This is simpler than general HTTP/2 case: not having an in-the-clear Upgrade path means the client is always in control of first HTTP/2 message, including any SETTINGS changes it may wish.

Additionally, the use of "prior knowledge" per section 3.4 of [RFC7540] is likely to also work particularly well in IoT scenarios in which a client and its web service are likely to be closely matched. In such scenarios, prior knowledge may allow for SETTINGS to be set in accordance with some shared state implied by the the prior knowledge. In such cases, SETTINGS negotiation may not be necessary in order to depart from the defaults as defined by HTTP/2.

7. Gateway and Proxying Issues

The proliferation of application and security protocols in the IoT has produced the deployments of islands of IoT devices, each using one of the several protocols available. However, usually an IoT deployment needs to communicate to another one, or at least needs to communicate with the Web, both because they have to upload data to the Cloud or because usually they are controlled by a Web application.

In such cases, communication is facilitated by a cross-protocol proxy or a gateway translating from one protocol syntax and semantic into another one. However, the presence of Cross-Protocol or Application gateways has at least two main drawbacks that need to be analyzed and addressed carefully.

- while the translation may be trivial for the basic scenarios, there are a lot of cases where the translation can lead to information loss or an incompatibility due to the different way different proxies make the translation.

- The presence of such devices may also become a critical point for security.

8. Implementation Considerations

This section assumes HTTP/2 over TCP.

In addition to underlying stack considerations with respect to IPv4, IPv6, TCP, and TLS, there are implementation considerations for HTTP/2 for IoT.

A primary consideration is the number of allowed simultaneous HTTP/2 connections. As each connection has associated overhead, as well as overhead for each stream, constrained hosts may wish to limit their number of simultaneous connections. However, implementers should consider that some popular browsers require more than one connection to operate.

In addition to minimizing the number of simultaneous connections, hosts should consider leaving connections open if there is a possibility of further communication with the remote peer. HTTP/2 contains mechanisms such as PING to periodically check idle connections. Leaving established connections open when there is a possibility of future communication allows connection establishment overhead (and potentially TLS session establishment overhead) to be avoided.

Should TLS be used, implementers may wish to consider utilizing hardware-based encryption to further reduce codespace and RAM requirements.

9. Experimentation and Performance

This section presents some simple results obtained using the Deuterium HTTP/2 library [Deuterium] and is not intended to be complete, but rather a start for discussion. From an IoT perspective, the reduced message sizes presented help to conserve both bandwidth and battery life, as well as potentially saving some memory/buffer space.

The results presented in this section make the following assumptions and considerations:

- Overhead from TCP and TLS are ignored
- An attempt to minimize the headers used has been made while still maintaining RFC compliance
- No entries are made into the HTTP/2 dynamic table, thus removing some potential optimization

- Connection establishment and teardown have been ignored, though clearly these are important considerations for IoT application protocols

- Only happy path transmissions are shown, thus no comparison of failure modes or retransmissions is given

9.1. GET Example

This first example compares and contrasts a GET method to a resource containing an XML representation of a simple switch using HTTP/1.1 and HTTP/2.

9.1.1. HTTP/1.1

1. Client sends (47 octets):

```
4745 5420 2f6f 6e6f 6666 2048 5454 502f 312e 310d
0a48 6f73 743a 2066 6f6f 0d0a 4163 6365 7074 3a20
2a2f 2a0d 0a0d 0a
```

In ASCII:

```
GET /onoff HTTP/1.1\r\n
Host: foo\r\n
Accept: */*\r\n
\r\n
```

2. Server sends (107 + 36 octets):

```
4854 5450 2f31 2e31 2032 3030 204f 4b0d 0a44 6174
653a 204d 6f6e 2c20 3039 204d 6172 2032 3031 3520
3036 3a32 363a 3434 2047 4d54 0d0a 436f 6e74 656e
742d 4c65 6e67 7468 3a20 3336 0d0a 436f 6e74 656e
742d 5479 7065 3a20 6170 706c 6963 6174 696f 6e2f
786d 6c0d 0a0d 0a
3c4f 6e4f 6666 3e0a 093c 7374 6174 653e 6f66 663c
2f73 7461 7465 3e0a 3c2f 4f6e 4f66 663e
```

In ASCII:

```
HTTP/1.1 200 OK\r\n
Date: Mon, 09 Mar 2015 06:26:44 GMT\r\n
Content-Length: 36\r\n
Content-Type: application/xml\r\n
\r\n
<OnOff>\n
\t<state>off</state>\n
</OnOff>
```

9.1.2. HTTP/2

1. Client sends (34 octets):

```
0000 1901 0500 0000 01
8286 0585 60f5 1e59 7f01 8294 e70f 0489 f963 e7ef
b401 5c00 07
```

Representing:

```
:method: GET
:path: /onoff
:scheme: http
:authority: foo
accept: */*
```

2. Server sends (54 octets):

```
0000 2d01 0400 0000 01
880f 1296 d07a be94 03ea 681d 8a08 016d 4039 704e
5c69 a531 68df 0f10 8b1d 75d0 620d 263d 4c79 a68f
0f0d 8265 cf
```

Representing:

```
:status: 200
content-type: application/xml
content-length: 36
date: Mon, 09 Mar 2015 06:26:44 GMT
```

3. Server sends (45 octets):

```
0000 2400 0100 0000 01
3c4f 6e4f 6666 3e0a 093c 7374 6174 653e 6f66 663c
2f73 7461 7465 3e0a 3c2f 4f6e 4f66 663e
```

Representing:

```
<OnOff>
\t<state>off</state>
</OnOff>
```

9.1.3. Comparison

In total and ignoring the payload (36 octets), the HTTP/2 flow is 37% smaller than the HTTP/1.1 flow.

The use of additional headers, particularly common headers that are present in the HTTP/2 static table, will result in greater savings.

While not compared here, HTTP/2's ability to reuse connections for multiple streams reduces connection establishment overhead, such as TCP connection establishment and TLS session establishment.

10. HTTP/2 over UDP - QUIC

QUIC (Quick UDP Internet Connections) is a new multiplexed transport protocol designed to run in user space above UDP, optimized for HTTP/2 semantics. In this document, "QUIC" refers to the upcoming IETF standard. The protocol is still in its early days and the standardization work in IETF has just started.

QUIC provides functionality already present in TCP and HTTP/2

- connection semantics, reliability, and congestion control equivalent to TCP.
- multiplexing and flow control equivalent to HTTP/2

Where functionality is similar to that of existing protocols, it has been re-designed to be more efficient. For example, the native multistream provides multiplexing without the head-of-line blocking inherent to HTTP/2 over TCP.

QUIC will use DTLS 1.3. Accordingly, connections will commonly benefit from 0-RTT as defined by TLS 1.3, meaning that on most QUIC connections, data can be sent immediately without waiting for a reply from the server. Furthermore, packets are always authenticated and typically the payload is fully encrypted.

QUIC has been designed to provide richer information to congestion control algorithms than TCP, moreover the actual congestion control is plugable in QUIC.

Even if QUIC has been initially designed with HTTP/2 as the primary application protocol to support, it is meant to become a modern

general-purpose transport protocol. The IETF standardization effort will also focus on describing the mapping of HTTP/2 semantics using QUIC specifically with the goal of minimizing web latency using QUIC. This mapping will accommodate the extension mechanisms defined in the HTTP/2 specification.

QUIC also dictates that packets should be sized to fit within the path's MTU to avoid IP fragmentation. However path MTU discovery is work in progress, and the current QUIC implementation uses a 1350-byte maximum QUIC packet size for IPv6, 1370 for IPv4.

Judging from its current state, QUIC may bring some potential benefits like the possibility to design and use a specific congestion control algorithm suited to IoT scenarios and possibility to reduce header overhead as compared to that of TCP plus HTTP/2. The latter is possible since these two layers are more integrated in QUIC.

11. IANA Considerations

This document has no considerations for IANA.

12. Security Considerations

Section 1 and Section 3 above point out security issues in the current IoT landscape, namely, the additional attack vectors from having several bespoke stacks instead of one mainstream stack and protocols. This document seeks to improve security of the IoT by encouraging use of mainstream protocols which are better understood and more thoroughly debugged (both in their specifications as well as in their implementations).

Section 7 point out another issue with the current IoT landscape: the proliferation of gateways and proxies. Whereas they serve useful functions in IoT, allowing more constrained nodes to have much lower duty cycles or filtering them from much traffic, there are inherent security issues, not the least of which is that they break end-to-end security. Enabling more mainstream protocols would not preclude using a proxy or gateway whenever the tradeoff dictated it, but would also allow for end-to-end security.

Given the security challenges in IoT scenarios, HTTP/2 is assumed to use TLS services. In Internet scenarios, [RFC7540] has clear guidance in this respect. In Constrained network scenarios, the guidance for IoT is [I-D.ietf-dice-profile]. However, these are currently at odds. For example, Section 4.2 of [I-D.ietf-dice-profile] mandates the ciphersuite TLS_PSK_WITH_AES_128_CCM_8 for preshared key-based authentication (quite common in IoT deployments). On the other hand, Appendix A of

[RFC7540] includes TLS_PSK_WITH_AES_128_CCM_8 in the HTTP/2 Black List of disallowed cipher suites, despite it being an AEAD ciphersuite. This is still to be resolved. The other IoT ciphersuite mandated by [I-D.ietf-dice-profile], namely, TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 (used for both certificate-based and Raw Public Key-based authentication) is not on the HTTP/2 Black List.

13. Acknowledgements

Thanks to the following individuals for helpful comments and discussion: Brian Raymor, Dave Thaler, Ed Briggs.

This document was produced using the xml2rfc tool [RFC2629][RFC7749].

14. References

14.1. Normative References

[I-D.ietf-dice-profile] Tschofenig, H. and T. Fossati, "TLS/DTLS Profiles for the Internet of Things", draft-ietf-dice-profile-17 (work in progress), October 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>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <http://www.rfc-editor.org/info/rfc7232>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <http://www.rfc-editor.org/info/rfc7233>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <http://www.rfc-editor.org/info/rfc7234>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <http://www.rfc-editor.org/info/rfc7235>.

[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>.

[RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <http://www.rfc-editor.org/info/rfc7541>.

14.2. Informative References

[Clapper] "US intelligence chief: we might use the internet of things to spy on you", February 2016, <http://www.theguardian.com/technology/2016/feb/09/internet-of-things-smart-home-devices-government-surveillance-james-clapper>.

[CoAP_integration] Giang, N., Ha, M., and D. Kim, "SCoAP: An integration of CoAP protocol with web-based application", Proc. IEEE GLOBECOM , 2013.

[Deuterium] Simpson, R., "Deuterium HTTP/2 Library", June 2016, <http://robbysimpson.org/deuterium/>.

[Eclipse_survey] Eclipse Foundation, "IoT Developer Survey", April 2016, <http://iot.ieee.org/images/files/pdf/iot-developer-survey-2016-report-final.pdf>.

[Going_Dark] "Dont Panic: Making Progress on Going Dark Debate", February 2016, <https://cyber.law.harvard.edu/pubrelease/dont-panic/Dont_Panic_Making_Progress_on_Going_Dark_Debate.pdf>.

[I-D.aggarwal-dnssd-optimize-query]
Aggarwal, A., "Optimizing DNS-SD query using TXT records", draft-aggarwal-dnssd-optimize-query-00 (work in progress), July 2014.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE)", draft-ietf-ace-oauth-authz-02 (work in progress), June 2016.

[IEEE_survey]
Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications", IEEE Communication Surveys and Tutorials , November 2015.

[IoT_analysis]
Colina, M., Bartolucci, M., Vanelli-Coralli, A., and G. Corazza, "Internet of Things application layer protocol analysis over error and delay prone links", Proc. ASMS/ SPSC Conference , 2014.

[mqtt_iso]
ISO, "ISO/IEC 20922:2016 Information technology -- Message Queuing Telemetry Transport (MQTT) v3.1.1", June 2016, <http://www.iso.org/iso/catalogue_detail.htm?csnumber=69466>.

[mqtt_oasis]
OASIS, "MQTT Version 3.1.1 becomes an OASIS Standard", October 2014, <https://www.oasis-open.org/news/announcements/mqtt-version-3-1-1-becomes-an-oasis-standard>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <http://www.rfc-editor.org/info/rfc2629>.

[RFC6762] Cheshire, S. and M. Krochmal, "Multicast DNS", RFC 6762, DOI 10.17487/RFC6762, February 2013, <http://www.rfc-editor.org/info/rfc6762>.

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <http://www.rfc-editor.org/info/rfc6763>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7452] Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", RFC 7452, DOI 10.17487/RFC7452, March 2015, <http://www.rfc-editor.org/info/rfc7452>.

[RFC7749] Reschke, J., "The "xml2rfc" Version 2 Vocabulary", RFC 7749, DOI 10.17487/RFC7749, February 2016, <http://www.rfc-editor.org/info/rfc7749>.

[Web_things] Lerche, C., Laum, N., Golatowski, F., Timmermann, D., and C. Niedermier, "Connecting the web with the web of things: lessons learned from implementing a CoAP-HTTP proxy", Proc. IEEE MASS , 2012.

Authors' Addresses

Gabriel Montenegro
Microsoft

Email: Gabriel.Montenegro@microsoft.com

Sandra Cespedes
NIC Chile Research Labs, Universidad de Chile

Email: scespedes@ing.uchile.cl

Salvatore Loreto
Ericsson

Email: salvatore.loreto@ericsson.com

Robby Simpson
General Electric

Email: rsimpson@gmail.com

httpbis
Internet-Draft
Intended status: Best Current Practice
Expires: May 4, 2017

D. Stenberg
Mozilla
T. Wicinski
Salesforce

October 31, 2016

TCP Tuning for HTTP
draft-stenberg-httpbis-tcp-03

Abstract

This document records current best practice for using all versions of HTTP over TCP.

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 May 4, 2017.

Table of Contents

1. Introduction 2
1.1. Notational Conventions 3
2. Socket planning 3
2.1. Number of open files 3
2.2. Number of concurrent network messages 3
2.3. Number of incoming TCP SYNs allowed to backlog 3
2.4. Use the whole port range for local ports 4
2.5. Lower the TCP FIN timeout 4
2.6. Reuse sockets in TIME_WAIT state 4
2.7. TCP socket buffer sizes and Window Scaling 4
2.8. Set maximum allowed TCP window sizes 5
2.9. Timers and timeouts 5
3. TCP handshake 5
3.1. TCP Fast Open 5
3.2. Initial Congestion Window 6
3.3. TCP SYN flood handling 6
4. TCP transfers 6
4.1. Packet Pacing 6
4.2. Explicit Congestion Control 6
4.3. Nagle's Algorithm 6
4.4. Delayed ACKs 7
4.5. Keep-alive 7
5. Re-using connections 8
5.1. Slow Start after Idle 8
5.2. TCP-Bound Authentications 8
6. Closing connections 8
6.1. Half-close 8
6.2. Abort 8
6.3. Close Idle Connections 8
6.4. Tail Loss Probes 9
7. IANA Considerations 9
8. Security Considerations 9
9. References 9
9.1. Normative References 9
9.2. Informative References 9
9.3. URIs 10
Appendix A. Acknowledgments 10
Appendix B. Operating System Settings for Linux 10
Authors' Addresses 12

1. Introduction

HTTP version 1.1 [RFC7230] as well as HTTP version 2 [RFC7540] are defined to use TCP [RFC0793], and their performance can depend greatly upon how TCP is configured. This document records the best

current practice for using HTTP over TCP, with a focus on improving end-user perceived performance.

These practices are generally applicable to HTTP/1 as well as HTTP/2, although some may note particular impact or nuance regarding a particular protocol version.

There are countless scenarios, roles and setups where HTTP is being using so there can be no single specific "Right Answer" to most TCP questions. This document intends only to cover the most important areas of concern and suggest possible actions.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Socket planning

Your HTTP server or intermediary may need configuration changes to some system tunables and timeout periods to perform optimally. Actual values will depend on how you are scaling the platform, horizontally or vertically, and other connection semantics. Changing system limits and altering thresholds will change the behavior of your web service and its dependencies. These dependencies are usually common to other services running on the same system, so good planning and testing is advised.

This is a list of values to consider and some general advice on how those values can be modified on Linux systems.

2.1. Number of open files

A modern HTTP server will serve a large number of TCP connections and in most systems each open socket equals an open file. Make sure that limit isn’t a bottle neck.

2.2. Number of concurrent network messages

Raise the number of packets allowed to get queued when a particular interface receives packets faster than the kernel can process them.

2.3. Number of incoming TCP SYNs allowed to backlog

The number of new connection requests that are allowed to queue up in the kernel. These can be connections that are in SYN RECEIVED or ESTABLISHED states. Historically, operating systems used a single

backlog queue for both of these states. Newer implemntations use two separate queues: one for connections in SYN RECEIVED and one for those which are ESTABLISHED state (better known as the accept queue).

2.4. Use the whole port range for local ports

To make sure the TCP stack can take full advantage of the entire set of possible sockets, give it a larger range of local port numbers to use.

2.5. Lower the TCP FIN timeout

High connection completion rates will consume ephemeral ports quickly. Lower the time during which connections are in FIN-WAIT-2/ TIME_WAIT states so that they can be purged faster and thus maintain a maximal number of available sockets. The primitives for the assignment of these values were described in [RFC0793], however significantly lower values are commonly used.

2.6. Reuse sockets in TIME_WAIT state

When running backend servers on a managed, low latency network you might allow the reuse of sockets in TIME_WAIT state for new connections when a protocol complete termination has occurred. There is no RFC that covers this behaviour.

2.7. TCP socket buffer sizes and Window Scaling

Systems meant to handle and serve a huge number of TCP connections at high speeds need a significant amount of memory for TCP socket buffers. On some systems you can tell the TCP stack what default buffer sizes to use and how much they are allowed to dynamically grow and shrink. Window Scaling is typically linked to socket buffer sizes.

The minimum and default tend to require less proactive amendment than the maximum value. When deriving maximum values for use, you should consider the BDP (Bandwidth Delay Product) of the target environment and clients. Consider also that ’read’ and ’write’ values do not require to be synchronised, as the BDP requirements for a load balancer or middle-box might be very different when acting as a sender or receiver.

Allowing needlessly high values beyond the expected limitations of the platform might increase the probability of retransmissions and buffer induced delays within the path. Extensions such as ECN coupled with AQM can help mitigate this undesirable behaviour [RFC7141].

[RFC7323] covers Window Scaling in greater detail.

2.8. Set maximum allowed TCP window sizes

You may have to increase the largest allowed window size. Window scaling must be accommodated within the maximal values, however it is not uncommon to see the maximum definable higher than the scalable limit; these values can statically defined within socket parameters (SO_RCVBUF,SO_SNDBUF).

2.9. Timers and timeouts

On a modern shared platform it can be common to plan for both long and short lived connections on the same implementation. However, the delivery of static assets and a 'web push' or 'long poll' service provide very different quality of service promises.

Fail 'fast': TCP resources can be highly contended. For fault tolerance reasons a server needs to be able to determine within a reasonable time frame whether a connection is still active or required. e.g. If static assets typically return in 100s of milliseconds, and users 'switch off' after <10s keeping timeouts of >30s make little sense and defining a 'quality of service' appropriate to the target platform is encouraged. On a shared platform with mixed session lifetimes, applications that require longer render times have various options to ensure the underlying service and upstream servers in the path can identify the session as not failed: HTTP continuations, Redirects, 202s or sending data.

Clients and servers typically have many timeout options, a few notable options are: Connect(client), time to request(server), time to first byte(client), between bytes(server/client), total connection time(server/client). Some implementations merge these values into a single 'timeout' definition even when statistics are reported individually. All should be considered as the defaults in many implementations are highly underiable, even infinite timeouts have been observed.

3. TCP handshake

3.1. TCP Fast Open

TCP Fast Open (a.k.a. TFO, [RFC7413]) allows data to be sent on the TCP handshake, thereby allowing a request to be sent without any delay if a connection is not open.

TFO requires both client and server support, and additionally requires application knowledge, because the data sent on the SYN

needs to be idempotent. Therefore, TFO can only be used on idempotent, safe HTTP methods (e.g., GET and HEAD), or with intervening negotiation (e.g, using TLS). It should be noted that TFO requires a secret to be defined on the server to mitigate security vulnerabilities it introduces. TFO therefore requires more server side deployment planning than other enhancements.

Support for TFO is growing in client platforms, especially mobile, due to the significant performance advantage it gives.

3.2. Initial Congestion Window

[RFC6928] specifies an initcwnd (initial congestion window) of 10, and is now fairly widely deployed server-side. There has been experimentation with larger initial windows, in combination with packet pacing. Many implementations allow initcwnd to be applied to specific routes which allows a greater degree of flexibility than some other TCP parameters.

IW10 has been reported to perform fairly well even in high volume servers.

3.3. TCP SYN flood handling

TCP SYN Flood mitigations [RFC4987] are necessary and there will be thresholds to tweak.

4. TCP transfers

4.1. Packet Pacing

TBD

4.2. Explicit Congestion Control

Apple deploying in iOS and OSX [1].

4.3. Nagle’s Algorithm

Nagle’s Algorithm [RFC0896] is the mechanism that makes the TCP stack hold (small) outgoing packets for a short period of time so that it can potentially merge that packet with the next outgoing one. It is optimized for throughput at the expense of latency.

HTTP/2 in particular requires that the client can send a packet back fast even during transfers that are perceived as single direction transfers. Even small delays in those sends can cause a significant performance loss.

HTTP/1.1 is also affected, especially when sending off a full request in a single write() system call.

In POSIX systems you switch it off like this:

```
int one = 1;
setsockopt(fd, IPPROTO_TCP, TCP_NODELAY, &one, sizeof(one));
```

4.4. Delayed ACKs

Delayed ACK [RFC1122] is a mechanism enabled in most TCP stacks that causes the stack to delay sending acknowledgement packets in response to data. The ACK is delayed up until a certain threshold, or until the peer has some data to send, in which case the ACK will be sent along with that data. Depending on the traffic flow and TCP stack this delay can be as long as 500ms.

This interacts poorly with peers that have Nagle's Algorithm enabled. Because Nagle's Algorithm delays sending until either one MSS of data is provided _or_ until an ACK is received for all sent data, delaying ACKs can force Nagle's Algorithm to buffer packets when it doesn't need to (that is, when the other peer has already processed the outstanding data).

Delayed ACKs can be useful in situations where it is reasonable to assume that a data packet will almost immediately (within 500ms) cause data to be sent in the other direction. In general in both HTTP/1.1 and HTTP/2 this is unlikely: therefore, disabling Delayed ACKs can provide an improvement in latency.

However, the TLS handshake is a clear exception to this case. For the duration of the TLS handshake it is likely to be useful to keep Delayed ACKs enabled.

Additionally, for low-latency servers that can guarantee responses to requests within 500ms, on long-running connections (such as HTTP/2), and when requests are small enough to fit within a small packet, leaving delayed ACKs turned on may provide minor performance benefits.

Effective use of switching off delayed ACKs requires extensive profiling.

4.5. Keep-alive

TCP keep-alive is likely disabled - at least on mobile clients for energy saving purposes. App-level keep-alive is then required for

long-lived requests to detect failed peers or connections reset by stateful firewalls etc.

5. Re-using connections

5.1. Slow Start after Idle

Slow-start is one of the algorithms that TCP uses to control congestion inside the network. It is also known as the exponential growth phase. Each TCP connection will start off in slow-start but will also go back to slow-start after a certain amount of idle time.

5.2. TCP-Bound Authentications

There are several HTTP authentication mechanisms in use today that are used or can be used to authenticate a connection rather than a single HTTP request. Two popular ones are NTLM and Negotiate.

If such an authentication has been negotiated on a TCP connection, that connection can remain authenticated throughout the rest of its lifetime. This discrepancy with how other HTTP authentications work makes it important to handle these connections with care.

6. Closing connections

6.1. Half-close

The client or server is free to half-close after a request or response has been completed; or when there is no pending stream in HTTP/2.

Half-closing is sometimes the only way for a server to make sure it closes down connections cleanly so that it doesn’t accept more requests while still allowing clients to receive the ongoing responses.

6.2. Abort

No client abort for HTTP/1.1 after the request body has been sent. Delayed full close is expected following an error response to avoid RST on the client.

6.3. Close Idle Connections

Keeping open connections around for subsequent connection reuse is key for many HTTP clients’ performance. The value of an existing connection quickly degrades and after only a few minutes the chance

that a connection will successfully get reused by a web browser is slim.

6.4. Tail Loss Probes

draft [2]

7. IANA Considerations

This document does not require action from IANA.

8. Security Considerations

TBD

9. References

9.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[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>.

[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>.

9.2. Informative References

[RFC0896] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <http://www.rfc-editor.org/info/rfc896>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <http://www.rfc-editor.org/info/rfc1122>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <http://www.rfc-editor.org/info/rfc4987>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <http://www.rfc-editor.org/info/rfc6928>.

[RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", BCP 41, RFC 7141, DOI 10.17487/RFC7141, February 2014, <http://www.rfc-editor.org/info/rfc7141>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <http://www.rfc-editor.org/info/rfc7323>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <http://www.rfc-editor.org/info/rfc7413>.

9.3. URIs

[1] https://developer.apple.com/videos/wwdc/2015/?id=719

[2] http://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe-01

Appendix A. Acknowledgments

This specification builds upon previous work and help from Mark Nottingham, Craig Taylor

Appendix B. Operating System Settings for Linux

Here are some sample operating system settings for the Linux operating system, along with the section it refers to.

Section 2.1

fs.file-max = <number of files>

Section 2.2

net.core.netdev_max_backlog = <number of packets>

Section 2.3

net.core.somaxconn = <number>

Section 2.4

net.ipv4.ip_local_port_range = 1024 65535

Section 2.5

net.ipv4.tcp_fin_timeout = <number of seconds>

Section 2.6

net.ipv4.tcp_tw_reuse = 1

Section 2.7

net.ipv4.tcp_wmem = <minimum size> <default size> <max size in bytes>

Section 2.7

net.ipv4.tcp_rmem = <minimum size> <default size> <max size in bytes>

Section 2.8

net.core.rmem_max = <number of bytes>

Section 2.8

net.core.wmem_max = <number of bytes>

Section 5.1

net.ipv4.tcp_slow_start_after_idle = 0

Section 4.3 Turning off Nagle’s Algorithm:

```
int one = 1;
setsockopt(fd, IPPROTO_TCP, TCP_NODELAY, &one, sizeof(one));
```

Section 4.4

On recent Linux kernels (since Linux 2.4.4), Delayed ACKs can be disabled like this:

```
int one = 1;
setsockopt(fd, IPPROTO_TCP, TCP_QUICKACK, &one, sizeof(one));
```

Unlike disabling Nagle's Algorithm, disabling Delayed ACKs on Linux is not a one-time operation: processing within the TCP stack can cause Delayed ACKs to be re-enabled. As a result, to use "TCP_QUICKACK" effectively requires setting and unsetting the socket option during the life of the connection.

Authors' Addresses

Daniel Stenberg
Mozilla

Email: daniel@haxx.se
URI: http://daniel.haxx.se

Tim Wicinski
Salesforce

Email: tjw.ietf@gmail.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 27, 2016

M. Thomson
C. Peterson
Mozilla

May 26, 2016

Expiring Aggressively Those HTTP Cookies
draft-thomson-http-omnomnom-00

Abstract

HTTP Cookies that are sent over connections without confidentiality and integrity protection are vulnerable to theft. Such cookies should be expired aggressively.

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 November 27, 2016.

Table of Contents

1. Introduction 2
1.1. Notational Conventions 2
2. Expire Cookies 2
3. Security Considerations 3
4. IANA Considerations 3
5. References 3
5.1. Normative References 3
5.2. Informative References 4
Appendix A. Acknowledgements 4
Authors' Addresses 4

1. Introduction

HTTP cookies [RFC6265] provide a means of persisting server state between multiple requests. This feature is widely used on both HTTP [RFC7230] and HTTPS [RFC2818] requests.

The authority for "http://" resources (see Section 9.1 of [RFC7230]) derives from insecure sources: notably the network and the DNS (absent DNSSEC). This situation might change over time. As persistent state, cookies create a way for an attacker to link requests. The information that a cookie holds might also be valuable to that attacker in some way.

To limit the effectiveness of attacks on cleartext communications [RFC7258], user agents are encouraged to limit the persistence of cookies that are set over insecure connections.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Expire Cookies

Cookies that are set using insecure channels (i.e., HTTP over cleartext TCP), MUST have a short time limit on the time that they are persisted. For instance, such cookies might only persist until the user closes their browser.

If a user agent detects a change in network conditions it SHOULD remove any cookies that were established using insecure channels.

Alternatives:

- In the investigation into this change, it was suggested that cookies without the "Secure" flag might be given the same treatment. However, this resulted in a far greater number of cookies being affected and some interoperability problems as a result.

- This change might be limited to cookies that are set in third-party contexts. See [I-D.west-first-party-cookies].

  Limiting access to third-party cookies in this fashion could have the secondary effect of encouraging providers of third-party content to move to HTTPS. This removes that content as a barrier to the adoption of HTTPS for the sites that include that content.

3. Security Considerations

This document describes an improvement that could be a security improvement. However, this is not without risks. For cookies that are used as a substitute for logins, more regular clearing of a login cookie could expose the primary authentication token (for instance, a password) to more network attackers as a result of being entered more often.

Clearing login tokens could also cause a degree of user annoyance, as login information is lost. Such annoyance manifests in many subtle ways.

Limiting the change to third-party contexts as suggested above might reduce these risks, though with lesser overall impact.

4. IANA Considerations

This document makes no request of IANA.

5. References

5.1. Normative References

[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>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <http://www.rfc-editor.org/info/rfc6265>.

[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>.

5.2. Informative References

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

Appendix A. Acknowledgements

Henri Sivonen first suggested that non-Secure cookies be made ephemeral. Chris Peterson did much of the initial investigation and work. See <https://bugzilla.mozilla.org/show_bug.cgi?id=1160368> for details.

Authors' Addresses

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

Chris Peterson
Mozilla

Email: cpeterson@mozilla.com

HTTPbis
Internet-Draft
Updates: 6265 (if approved)
Intended status: Standards Track
Expires: September 4, 2016

E. Wright
Shopify
S. Huang
M. West
Google, Inc

March 3, 2016

A Retention Priority Attribute for HTTP Cookies
draft-west-cookie-priority-00

Abstract

This document defines the "Priority" attribute for HTTP cookies. This attribute allows servers to specify a retention priority for HTTP cookies that will be respected by user agents during cookie eviction.

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 4, 2016.

Copyright Notice

Table of Contents

1. Introduction 2
2. Terminology and notation 2
3. Overview 3
3.1. Examples 3
4. Server Requirements 4
4.1. Syntax 4
4.2. Semantics (Non-Normative) 5
4.3. The 'Priority' Attribute 5
5. User Agent Requirements 5
5.1. The 'Priority' Attribute 5
5.2. Storage Model 5
6. Implementation Considerations 6
7. References 6
7.1. Normative References 6
7.2. Informative References 7
Appendix A. Acknowledgements 7
Authors' Addresses 7

1. Introduction

This document defines the "Priority" attribute for HTTP cookies. Using the "Priority" attribute, servers may indicate that certain cookies should be protected, and others preferentially deleted. When a user agent evicts cookies in the enforcement of a per-domain quota, lower priority cookies will be deleted first, potentially preserving higher-priority cookies that would otherwise have been deleted according to the rules of [RFC6265].

2. Terminology and notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].

Two sequences of octets are said to case-insensitively match each other if and only if they are equivalent under the "i;ascii-casemap" collation defined in [RFC4790].

3. Overview

This section outlines a way for an origin server to indicate the retention priority for individual cookies and for the user agent to respect retention priorities during cookie eviction.

To indicate a cookie’s retention priority, the origin server includes a "Priority" attribute in the "Set-Cookie" HTTP response header. A value of "Low" indicates that the cookie should be given lower retention priority (evicted prior to other cookies). A value of "High" indicates that a cookie should be given higher retention priority (evicted after other cookies). The value of "Medium" corresponds to the default behavior.

In order to prevent starvation of functionality dependent on low- and medium-priority cookies, a fraction of the cookie quota should be reserved for them.

[RFC6265], Section 5.3, describes a strategy that user agents must follow when "removing excess cookies". A user agent implementing the current specification for a retention priority attribute will implement an extended priority order, dividing the second priority ("Cookies that share a domain field with more than a predetermined number of other cookies") into multiple levels corresponding to the three retention priorities.

As a result, when the cookies for a given domain exceed user agent limits, cookies with low priority will be evicted first, followed by medium and high priority cookies.

3.1. Examples

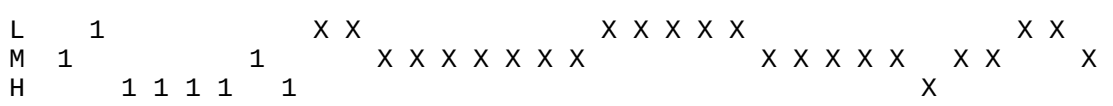
Using the Priority attribute, an origin server may assign a retention priority to a cookie stored by a user agent. For example, the origin server may prioritize the retention of session security tokens while indicating that superficial data such as a user’s favorite color should be discarded in preference to other data.

The following figure illustrates a series of 33 cookies received by a user-agent from the example.com server during one or more HTTP responses. Each cookie is represented by an L, M, or H indicating low, medium, or high priority, respectively.

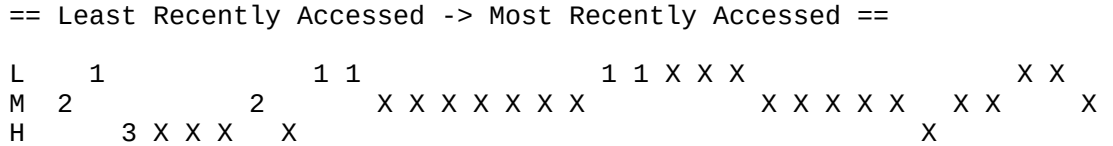
```
== Least Recently Accessed -> Most Recently Accessed ==

   M L H H H H M H L L M M M M M M M L L L L L M M M M M H M M L L M
```

Assume that the user agent is configured to evict all but 25 cookies from a domain when the number of cookies exceeds 31. [RFC6265] specifies the following eviction order. Cookies are separated in the vertical axis by priority. Evicted cookies are labeled with a '1' and retained cookies with an 'X'.



A user agent that implements the current specification, reserving room for 5 low-, 15 medium-, and 5 high-priority cookies, would implement a modified eviction order as follows. '1', '2', and '3' indicate cookies evicted during various phases of the algorithm.



Of note is that the retention priority does not impact the relative eviction priority of cookies being evicted due to the global threshold (i.e., once no domain exceeds the per-domain threshold). Furthermore, this new attribute has no effect on domains that do not send it.

4. Server Requirements

This section describes the syntax and semantics of the "Priority" cookie attribute.

4.1. Syntax

The Set-Cookie HTTP response header syntax is defined in [RFC6265], Section 4.1.1. The grammar defined therein provides for tokens of type 'extension-av'. The Priority attribute is a subset of 'extension-av' that may appear zero or one times in a given 'set-cookie-header'. If it appears, the 'priority' attribute must conform to the following grammar:

```
priority-av    = "Priority=" priority-value
priority-value = "Low" / "Medium" / "High"
```

4.2. Semantics (Non-Normative)

This section describes a simplified semantics of the "Priority" attribute in the "Set-Cookie" HTTP response header. The full semantics are described in Section 5.

4.3. The 'Priority' Attribute

The "Priority" attribute indicates a retention priority relative to other cookies from the same domain as the cookie carrying the attribute. During cookie eviction in enforcement of per-domain cookie limits, "Low" priority cookies will be evicted before "Medium" and "Medium" before "High". Cookies without a specified priority are considered to have "Medium" priority.

5. User Agent Requirements

[RFC6265], Section 5.2 describes how user agents must parse the value of the "Set-Cookie" HTTP response header. This specification provides additional processing steps that user agents must follow when they encounter a "Priority" attribute.

"Set-Cookie" headers that do not specify the "Priority" attribute MUST be treated as if the attribute was present and had the value Medium.

5.1. The 'Priority' Attribute

If the "attribute-name" case-insensitively matches the string "Priority", the user agent MUST process the "cookie-av" as follows:

If the "attribute-value" case-insensitively matches the string "Low", the cookie is assigned a low retention priority.

If the "attribute-value" case-insensitively matches the string "Medium", the cookie is assigned a medium retention priority.

If the "attribute-value" case-insensitively matches the string "High", the cookie is assigned a high retention priority.

Otherwise, the cookie is assigned a medium retention priority.

5.2. Storage Model

[RFC6265], Section 6.1 recommends that user agents set limits on the number of cookies they will store. [RFC6265], Section 5.3, describes a strategy that user must follow when "removing excess cookies".

A user agent implementing the current specification for a retention priority attribute will set aside a small portion of its storage quota for low-priority cookies, and another portion for medium-priority cookies. During eviction, compatible user agents will implement an extended priority order, dividing the second priority ("Cookies that share a domain field with more than a predetermined number of other cookies") into three, according to the retention priorities and the space reserved for them. Assuming that no other extensions ammend the rules defined in [RFC6265], a compatible user agent MUST therefore evict cookies in the following priority order (L and M refer to the amount of space reserved for low- and medium-priority cookies respectively, per domain). As per [RFC6265], within each category the least-recently accessed cookies should be deleted first.

1. Expired cookies.

2. Cookies with a low retention priority that share a domain field with more than a predetermined number of other cookies, excluding the first L low-priority cookies from that domain.

3. Cookies with a low or medium retention priority that share a domain field with more than a predetermined number of other cookies, excluding the first L + M low- and medium-priority cookies from that domain.

4. Cookies that share a domain field with more than a predetermined number of other cookies.

5. All cookies.

6. Implementation Considerations

This specification extends [RFC6265] in order to enable improved behaviour when servers are unable or unwilling to keep the number of distinct cookies served by their domains within the limits of user agents. As this specification is unlikely to ever achieve universal adoption by user agents, servers SHOULD gracefully degrade if their specifed cookie retention priorities are not respected.

7. References

7.1. Normative References

[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>.

[RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, DOI 10.17487/RFC4790, March 2007, <http://www.rfc-editor.org/info/rfc4790>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <http://www.rfc-editor.org/info/rfc6265>.

7.2. Informative References

[Wright2013] Wright, E. and S. Huang, "A Retention Priority Attribute for HTTP Cookies", n.d., <https://docs.google.com/a/google.com/file/d/0B3o1IlTKoADVRllKWGlyWGxIVTg/edit>.

Appendix A. Acknowledgements

This document is based heavily on an earlier draft written by Erik Wright and Samuel Huang [Wright2013], and the experimentation done in cooperation with Google’s login team.

Authors’ Addresses

Erik Wright
Shopify

Samuel Huang
Google, Inc

Email: huangs@google.com

Mike West
Google, Inc

Email: mkwst@google.com

HTTPbis
Internet-Draft
Updates: 6265 (if approved)
Intended status: Standards Track
Expires: October 8, 2016

M. West
Google, Inc
M. Goodwin
Mozilla

April 6, 2016

Same-site Cookies
draft-west-first-party-cookies-07

Abstract

This document updates RFC6265 by defining a "SameSite" attribute which allows servers to assert that a cookie ought not to be sent along with cross-site requests. This assertion allows user agents to mitigate the risk of cross-origin information leakage, and provides some protection against cross-site request forgery attacks.

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 October 8, 2016.

Table of Contents

1. Introduction 2
1.1. Goals 3
1.2. Examples 3
2. Terminology and notation 4
2.1. "Same-site" and "cross-site" Requests 4
2.1.1. Document-based requests 5
2.1.2. Worker-based requests 6
3. Server Requirements 7
3.1. Grammar 7
3.2. Semantics of the "SameSite" Attribute (Non-Normative) . . 8
4. User Agent Requirements 8
4.1. The "SameSite" attribute 8
4.1.1. "Strict" and "Lax" enforcement 8
4.2. Monkey-patching the Storage Model 9
4.3. Monkey-patching the "Cookie" header 10
5. Authoring Considerations 10
5.1. Defense in depth 10
5.2. Top-level Navigations 11
5.3. Mashups and Widgets 11
6. Privacy Considerations 11
6.1. Server-controlled 11
6.2. Pervasive Monitoring 12
7. References 12
7.1. Normative References 12
7.2. Informative References 13
Appendix A. Acknowledgements 14
Authors' Addresses 14

1. Introduction

Section 8.2 of [RFC6265] eloquently notes that cookies are a form of ambient authority, attached by default to requests the user agent sends on a user's behalf. Even when an attacker doesn't know the contents of a user's cookies, she can still execute commands on the user's behalf (and with the user's authority) by asking the user agent to send HTTP requests to unwary servers.

Here, we update [RFC6265] with a simple mitigation strategy that allows servers to declare certain cookies as "same-site", meaning they should not be attached to "cross-site" requests (as defined in section 2.1).

Note that the mechanism outlined here is backwards compatible with the existing cookie syntax. Servers may serve these cookies to all user agents; those that do not support the "SameSite" attribute will simply store a cookie which is attached to all relevant requests, just as they do today.

1.1. Goals

These cookies are intended to provide a solid layer of defense-in-depth against attacks which require embedding an authenticated request into an attacker-controlled context:

1. Timing attacks which yield cross-origin information leakage (such as those detailed in [pixel-perfect]) can be substantially mitigated by setting the "SameSite" attribute on authentication cookies. The attacker will only be able to embed unauthenticated resources, as embedding mechanisms such as "<iframe>" will yield cross-site requests.

2. Cross-site script inclusion (XSSI) attacks are likewise mitigated by setting the "SameSite" attribute on authentication cookies. The attacker will not be able to include authenticated resources via "<script>" or "<link>", as these embedding mechanisms will likewise yield cross-site requests.

3. Cross-site request forgery (CSRF) attacks which rely on top-level navigation (HTML "<form>" POSTs, for instance) can also be mitigated by treating these navigational requests as "cross-site".

4. Same-site cookies have some marginal value for policy or regulatory purposes, as cookies which are not delivered with cross-site requests cannot be directly used for tracking purposes. It may be valuable for an origin to assert that its cookies should not be sent along with cross-site requests in order to limit its exposure to non-technical risk.

1.2. Examples

Same-site cookies are set via the "SameSite" attribute in the "Set-Cookie" header field. That is, given a server’s response to a user agent which contains the following header field:

```
Set-Cookie: SID=31d4d96e407aad42; SameSite=Strict
```

Subsequent requests from that user agent can be expected to contain the following header field if and only if both the requested resource and the resource in the top-level browsing context match the cookie.

2. Terminology and notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234].

Two sequences of octets are said to case-insensitively match each other if and only if they are equivalent under the "i;ascii-casemap" collation defined in [RFC4790].

The terms "active document", "ancestor browsing context", "browsing context", "document", "WorkerGlobalScope", "sandboxed origin browsing context flag", "parent browsing context", "the worker's Documents", "nested browsing context", and "top-level browsing context" are defined in [HTML].

"Service Workers" are defined in the Service Workers specification [SERVICE-WORKERS].

The term "origin", the mechanism of deriving an origin from a URI, and the "the same" matching algorithm for origins are defined in [RFC6454].

"Safe" HTTP methods include "GET", "HEAD", "OPTIONS", and "TRACE", as defined in Section 4.2.1 of [RFC7231].

The term "public suffix" is defined in a note in Section 5.3 of [RFC6265] as "a domain that is controlled by a public registry". For example, "example.com"'s public suffix is "com". User agents SHOULD use an up-to-date public suffix list, such as the one maintained by Mozilla at [PSL].

An origin's "registrable domain" is the origin's host's public suffix plus the label to its left. That is, "https://www.example.com"'s registrable domain is "example.com". This concept is defined more rigorously in [PSL].

The term "request", as well as a request's "client", "current url", "method", and "target browsing context", are defined in [FETCH].

2.1. "Same-site" and "cross-site" Requests

A request is "same-site" if its target's URI's origin's registrable domain is an exact match for the request's initiator's "site for cookies", and "cross-site" otherwise. To be more precise, for a

given request ("request"), the following algorithm returns "same-site" or "cross-site":

1. If "request"'s client is "null", return "same-site".

2. Let "site" be "request"'s client's "site for cookies" (as defined in the following sections).

3. Let "target" be the registrable domain of "request"'s current url.

4. If "site" is an exact match for "target", return "same-site".

5. Return "cross-site".

2.1.1. Document-based requests

The URI displayed in a user agent's address bar is the only security context directly exposed to users, and therefore the only signal users can reasonably rely upon to determine whether or not they trust a particular website. The registrable domain of that URI's origin represents the context in which a user most likely believes themselves to be interacting. We'll label this domain the "top-level site".

For a document displayed in a top-level browsing context, we can stop here: the document's "site for cookies" is the top-level site.

For documents which are displayed in nested browsing contexts, we need to audit the origins of each of a document's ancestor browsing contexts' active documents in order to account for the "multiple-nested scenarios" described in Section 4 of [RFC7034]. These document's "site for cookies" is the top-level site if and only if the document and each of its ancestor documents' origins have the same registrable domain as the top-level site. Otherwise its "site for cookies" is the empty string.

Given a Document ("document"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Let "top-document" be the active document in "document"'s browsing context's top-level browsing context.

2. Let "top-origin" be the origin of "top-document"'s URI if "top-document"'s sandboxed origin browsing context flag is set, and "top-document"'s origin otherwise.

3. Let "documents" be a list containing "document" and each of "document"'s ancestor browsing contexts' active documents.

4. For each "item" in "documents":

    1. Let "origin" be the origin of "item"'s URI if "item"'s sandboxed origin browsing context flag is set, and "item"'s origin otherwise.

    2. If "origin"'s host's registrable domain is not an exact match for "top-origin"'s host's registrable domain, return the empty string.

5. Return "top-site".

2.1.2. Worker-based requests

Worker-driven requests aren't as clear-cut as document-driven requests, as there isn't a clear link between a top-level browsing context and a worker. This is especially true for Service Workers [SERVICE-WORKERS], which may execute code in the background, without any document visible at all.

Note: The descriptions below assume that workers must be same-origin with the documents that instantiate them. If this invariant changes, we'll need to take the worker's script's URI into account when determining their status.

2.1.2.1. Dedicated and Shared Workers

Dedicated workers are simple, as each dedicated worker is bound to one and only one document. Requests generated from a dedicated worker (via "importScripts", "XMLHttpRequest", "fetch()", etc) define their "site for cookies" as that document's "site for cookies".

Shared workers may be bound to multiple documents at once. As it is quite possible for those documents to have distinct "site for cookie" values, the worker's "site for cookies" will be the empty string in cases where the values diverge, and the shared value in cases where the values agree.

Given a WorkerGlobalScope ("worker"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Let "site" be "worker"'s origin's host's registrable domain.

2. For each "document" in "worker"'s Documents:

1. Let "document-site" be "document"'s "site for cookies" (as defined in Section 2.1.1).

2. If "document-site" is not an exact match for "site", return the empty string.

3. Return "site".

2.1.2.2. Service Workers

Service Workers are more complicated, as they act as a completely separate execution context with only tangential relationship to the Document which registered them.

Requests which simply pass through a service worker will be handled as described above: the request's client will be the Document or Worker which initiated the request, and its "site for cookies" will be those defined in Section 2.1.1 and Section 2.1.2.1

Requests which are initiated by the Service Worker itself (via a direct call to "fetch()", for instance), on the other hand, will have a client which is a ServiceWorkerGlobalScope. Its "site for cookies" will be the registrable domain of the Service Worker's URI.

Given a ServiceWorkerGlobalScope ("worker"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Return "worker"'s origin's host's registrable domain.

3. Server Requirements

This section describes extensions to [RFC6265] necessary to implement the server-side requirements of the "SameSite" attribute.

3.1. Grammar

Add "SameSite" to the list of accepted attributes in the "Set-Cookie" header field's value by replacing the "cookie-av" token definition in Section 4.1.1 of [RFC6265] with the following ABNF grammar:

```
cookie-av      = expires-av / max-age-av / domain-av /
                 path-av / secure-av / httponly-av /
                 samesite-av / extension-av
samesite-av    = "SameSite" / "SameSite=" samesite-value
samesite-value = "Strict" / "Lax"
```

3.2. Semantics of the "SameSite" Attribute (Non-Normative)

The "SameSite" attribute limits the scope of the cookie such that it will only be attached to requests if those requests are "same-site", as defined by the algorithm in Section 2.1. For example, requests for "https://example.com/sekrit-image" will attach same-site cookies if and only if initiated from a context whose "site for cookies" is "example.com".

If the "SameSite" attribute's value is "Strict", or if the value is invalid, the cookie will only be sent along with "same-site" requests. If the value is "Lax", the cookie will be sent with "same-site" requests, and with "cross-site" top-level navigations, as described in Section 4.1.1.

The changes to the "Cookie" header field suggested in Section 4.3 provide additional detail.

4. User Agent Requirements

This section describes extensions to [RFC6265] necessary in order to implement the client-side requirements of the "SameSite" attribute.

4.1. The "SameSite" attribute

The following attribute definition should be considered part of the the "Set-Cookie" algorithm as described in Section 5.2 of [RFC6265]:

If the "attribute-name" case-insensitively matches the string "SameSite", the user agent MUST process the "cookie-av" as follows:

1. If "cookie-av"'s "attribute-value" is not a case-sensitive match for "Strict" or "Lax", ignore the "cookie-av".

2. Let "enforcement" be "Lax" if "cookie-av"'s "attribute-value" is a case-insensitive match for "Lax", and "Strict" otherwise.

3. Append an attribute to the "cookie-attribute-list" with an "attribute-name" of "SameSite" and an "attribute-value" of "enforcement".

4.1.1. "Strict" and "Lax" enforcement

By default, same-site cookies will not be sent along with top-level navigations. As discussed in Section 5.2, this might or might not be compatible with existing session management systems. In the interests of providing a drop-in mechanism that mitigates the risk of CSRF attacks, developers may set the "SameSite" attribute in a "Lax"

enforcement mode that carves out an exception which sends same-site cookies along with cross-site requests if and only if they are top-level navigations which use a "safe" (in the [RFC7231] sense) HTTP method.

Lax enforcement provides reasonable defense in depth against CSRF attacks that rely on unsafe HTTP methods (like "POST"), but do not offer a robust defense against CSRF as a general category of attack:

1. Attackers can still pop up new windows or trigger top-level navigations in order to create a "same-site" request (as described in section 2.1), which is only a speedbump along the road to exploitation.

2. Features like "<link rel='prerender'>" [prerendering] can be exploited to create "same-site" requests without the risk of user detection.

When possible, developers should use a session management mechanism such as that described in Section 5.2 to mitigate the risk of CSRF more completely.

4.2. Monkey-patching the Storage Model

Note: There's got to be a better way to specify this. Until I figure out what that is, monkey-patching!

Alter Section 5.3 of [RFC6265] as follows:

1. Add "samesite-flag" to the list of fields stored for each cookie. This field's value is one of "None", "Strict", or "Lax".

2. Before step 11 of the current algorithm, add the following:

    1. If the "cookie-attribute-list" contains an attribute with an "attribute-name" of "SameSite", set the cookie's "samesite-flag" to "attribute-value" ("Strict" or "Lax"). Otherwise, set the cookie's "samesite-flag" to "None".

    2. If the cookie's "samesite-flag" is not "None", and the request which generated the cookie's client's "site for cookies" is not an exact match for "request-uri"'s host's registrable domain, then abort these steps and ignore the newly created cookie entirely.

4.3. Monkey-patching the "Cookie" header

Note: There's got to be a better way to specify this. Until I figure out what that is, monkey-patching!

Alter Section 5.4 of [RFC6265] as follows:

1. Add the following requirement to the list in step 1:

    * If the cookie's "samesite-flag" is not "None", and the HTTP request is cross-site (as defined in Section 2.1 then exclude the cookie unless all of the following statements hold:

        1. "samesite-flag" is "Lax"

        2. The HTTP request's method is "safe".

        3. The HTTP request's target browsing context is a top-level browsing context.

Note that the modifications suggested here concern themselves only with the "site for cookies" of the request's client, and the registrable domain of the resource being requested. The cookie's "domain", "path", and "secure" attributes do not come into play for these comparisons.

5. Authoring Considerations

5.1. Defense in depth

"SameSite" cookies offer a robust defense against CSRF attack when deployed in strict mode, and when supported by the client. It is, however, prudent to ensure that this designation is not the extent of a site's defense against CSRF, as same-site navigations and submissions can certainly be executed in conjunction with other attack vectors such as cross-site scripting.

Developers are strongly encouraged to deploy the usual server-side defenses (CSRF tokens, ensuring that "safe" HTTP methods are idempotent, etc) to mitigate the risk more fully.

Additionally, client-side techniques such as those described in [app-isolation] may also prove effective against CSRF, and are certainly worth exploring in combination with "SameSite" cookies.

5.2. Top-level Navigations

Setting the "SameSite" attribute in "strict" mode provides robust defense in depth against CSRF attacks, but has the potential to confuse users unless sites' developers carefully ensure that their session management systems deal reasonably well with top-level navigations.

Consider the scenario in which a user reads their email at MegaCorp Inc's webmail provider "https://example.com/". They might expect that clicking on an emailed link to "https://projects.com/secret/project" would show them the secret project that they're authorized to see, but if "projects.com" has marked their session cookies as "SameSite", then this cross-site navigation won't send them along with the request. "projects.com" will render a 404 error to avoid leaking secret information, and the user will be quite confused.

Developers can avoid this confusion by adopting a session management system that relies on not one, but two cookies: one conceptualy granting "read" access, another granting "write" access. The latter could be marked as "SameSite", and its absence would provide a reauthentication step before executing any non-idempotent action. The former could drop the "SameSite" attribute entirely, or choose the "Lax" version of enforcement, in order to allow users access to data via top-level navigation.

5.3. Mashups and Widgets

The "SameSite" attribute is inappropriate for some important use-cases. In particular, note that content intended for embedding in a cross-site contexts (social networking widgets or commenting services, for instance) will not have access to such cookies. Cross-site cookies may be required in order to provide seamless functionality that relies on a user's state.

Likewise, some forms of Single-Sign-On might require authentication in a cross-site context; these mechanisms will not function as intended with same-site cookies.

6. Privacy Considerations

6.1. Server-controlled

Same-site cookies in and of themselves don't do anything to address the general privacy concerns outlined in Section 7.1 of [RFC6265]. The attribute is set by the server, and serves to mitigate the risk of certain kinds of attacks that the server is worried about. The user is not involved in this decision. Moreover, a number of side-

channels exist which could allow a server to link distinct requests even in the absence of cookies. Connection and/or socket pooling, Token Binding, and Channel ID all offer explicit methods of identification that servers could take advantage of.

6.2. Pervasive Monitoring

As outlined in [RFC7258], pervasive monitoring is an attack. Cookies play a large part in enabling such monitoring, as they are responsible for maintaining state in HTTP connections. We considered restricting same-site cookies to secure contexts [secure-contexts] as a mitigation but decided against doing so, as this feature should result in a strict reduction in the number of cookies floating around in cross-site contexts. That is, even if "http://not-example.com" embeds a resource from "http://example.com/", that resource will not be "same-site", and "http://example.com"'s cookies simply cannot be used to correlate user behavior across distinct origins.

7. References

7.1. Normative References

[FETCH] van Kesteren, A., "Fetch", n.d., <https://fetch.spec.whatwg.org/>.

[HTML] Hickson, I., Pieters, S., van Kesteren, A., Jaegenstedt, P., and D. Denicola, "HTML", n.d., <https://html.spec.whatwg.org/>.

[PSL] "Public Suffix List", n.d., <https://publicsuffix.org/list/>.

[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>.

[RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, DOI 10.17487/RFC4790, March 2007, <http://www.rfc-editor.org/info/rfc4790>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <http://www.rfc-editor.org/info/rfc6265>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <http://www.rfc-editor.org/info/rfc6454>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[SERVICE-WORKERS] Russell, A., Song, J., and J. Archibald, "Service Workers", n.d., <http://www.w3.org/TR/service-workers/>.

7.2. Informative References

[app-isolation] Chen, E., Bau, J., Reis, C., Barth, A., and C. Jackson, "App Isolation - Get the Security of Multiple Browsers with Just One", n.d., <http://www.collinjackson.com/research/papers/appisolation.pdf>.

[pixel-perfect] Stone, P., "Pixel Perfect Timing Attacks with HTML5", n.d., <http://www.contextis.com/documents/2/Browser_Timing_Attacks.pdf>.

[prerendering] Bentzel, C., "Chrome Prerendering", n.d., <https://www.chromium.org/developers/design-documents/prerender>.

[RFC7034] Ross, D. and T. Gondrom, "HTTP Header Field X-Frame-Options", RFC 7034, DOI 10.17487/RFC7034, October 2013, <http://www.rfc-editor.org/info/rfc7034>.

[samedomain-cookies] Goodwin, M. and J. Walker, "SameDomain Cookie Flag", 2011, <http://people.mozilla.org/~mgoodwin/SameDomain/samedomain-latest.txt>.

[secure-contexts]
West, M., "Secure Contexts", n.d., <https://w3c.github.io/webappsec-secure-contexts/>.

Appendix A. Acknowledgements

The same-site cookie concept documented here is indebited to Mark Goodwin's and Joe Walker's [samedomain-cookies]. Michal Zalewski, Artur Janc, Ryan Sleevi, and Adam Barth provided particularly valuable feedback on this document.

Authors' Addresses

Mike West
Google, Inc

Email: mkwst@google.com
URI: https://mikewest.org/

Mark Goodwin
Mozilla

Email: mgoodwin@mozilla.com
URI: https://www.computerist.org/