

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/>.

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 3, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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.

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

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

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

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

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

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

1.2.2. HTTP/1.1 using TLS 1.3

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

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

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

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

1.2.3. HTTP/2

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

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

Thus, the minimum necessary mechanism to support reactive certificate authentication in HTTP/2 is an identifier that can be used 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.

```

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

```

Figure 3: Proactive Server Certificate

```

Client                                     Server
-- (stream 0) CERTIFICATE (AU flag) ----->
-- (streams 1,3) GET /protected ----->
<----- (streams 1,3) 200 OK --

```

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.

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

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.

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

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".

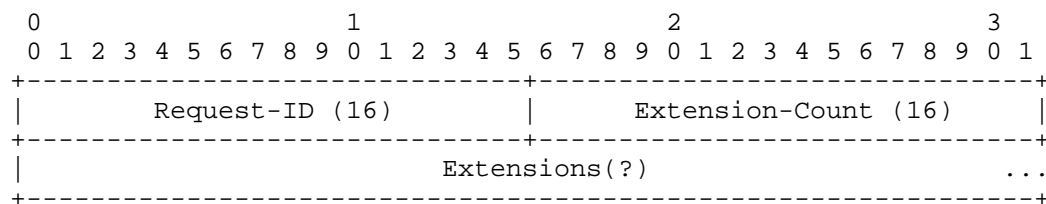


Figure 7: CERTIFICATE_REQUEST frame payload

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.

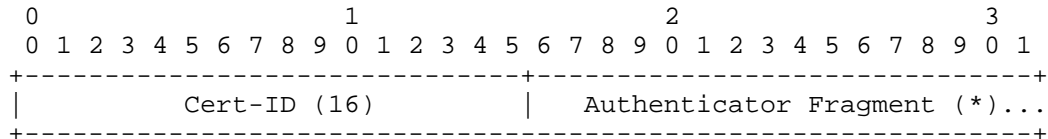


Figure 8: CERTIFICATE frame payload

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:

- o the "validate" API confirms the validity of the authenticator itself
- o 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 is 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/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

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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).

```

+-----+-----+
|          Origin-Len (16)          | Origin? (\*)          ...
+-----+-----+
|                      Digest-Value? (\*)                      ...
+-----+-----+
```

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:

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

- o ***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:

- o "P", an integer smaller than 256, that indicates the probability of a false positive that is acceptable, expressed as "1/2**P".
- o "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 $f * \text{"allocated"} * b / 8$ 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:

- o "URL" a string corresponding to the Effective Request URI ([RFC7230], Section 5.5) of a cached response [RFC7234]
 - o "maxcount" - max number of cuckoo hops
 - o "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:

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

- o "key", an array of characters
 - o "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 infinitesimally 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:

- o "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:

- o "key", an array of characters.
- o "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:

- o "hash1", an integer indicating the previous hash.
- o "fingerprint", an integer indicating the fingerprint value.
- o "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;

- o 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.
- o 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:

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

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

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

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

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

- o Code: 0x7
- o Name: SETTINGS_ACCEPT_CACHE_DIGEST
- o Initial Value: 0x0
- o 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 represents 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

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

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

- o Yoav becomes an author; Mark steps down.

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

- o Switch to Cuckoo Filter.

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

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

- o Change intended status from Standard to Experimental.

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

- o 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 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 January 4, 2021.

Copyright Notice

Copyright (c) 2020 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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:

- o 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.
- o Cookie-based approaches are not portable across applications and servers, impose additional client-side latency by requiring JavaScript execution, and are not cache friendly.
- o 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:

- o 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.

- o 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.
- o 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:

- o 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.
- o 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.
- o 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:

- o Potential conflicts due to existing use of header field name
- o 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

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

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

- o Issue 153 (NetInfo API citation) added normative reference.
- A.2. Since -01
- o Issue 200: Moved Key reference to informative.
 - o Issue 215: Extended passive fingerprinting and mitigation considerations.
 - o Changed document status to experimental.
- A.3. Since -02
- o Issue 239: Updated reference to CR-css-values-3
 - o Issue 240: Updated reference for Network Information API
 - o Issue 241: Consistency in IANA considerations
 - o Issue 250: Clarified Accept-CH
- A.4. Since -03
- o Issue 284: Extended guidance for Accept-CH
 - o Issue 308: Editorial cleanup
 - o Issue 306: Define Accept-CH-Lifetime
- A.5. Since -04
- o Issue 361: Removed Downlink
 - o Issue 361: Moved Key to appendix, plus other editorial feedback
- A.6. Since -05
- o Issue 372: Scoped CH opt-in and delivery to secure transports
 - o Issue 373: Bind CH opt-in to origin
- A.7. Since -06
- o Issue 524: Save-Data is now defined by NetInfo spec, dropping
 - o PR 775: Removed specific features to be defined in other specifications
- A.8. Since -07
- o Issue 761: Clarified that the defined headers are response headers.
 - o Issue 730: Replaced Key reference with Variants.
 - o Issue 700: Replaced ABNF with structured headers.
 - o PR 878: Removed Accept-CH-Lifetime based on feedback at IETF 105

A.9. Since -08

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

A.10. Since -09

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

A.11. Since -10

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

A.12. Since -11

- o PR 1134: Re-insert back section.

A.13. Since -12

- o PR 1160: AD review.

A.14. Since -13

- o PR 1171: Genart review.

A.15. Since -14

- o 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
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.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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

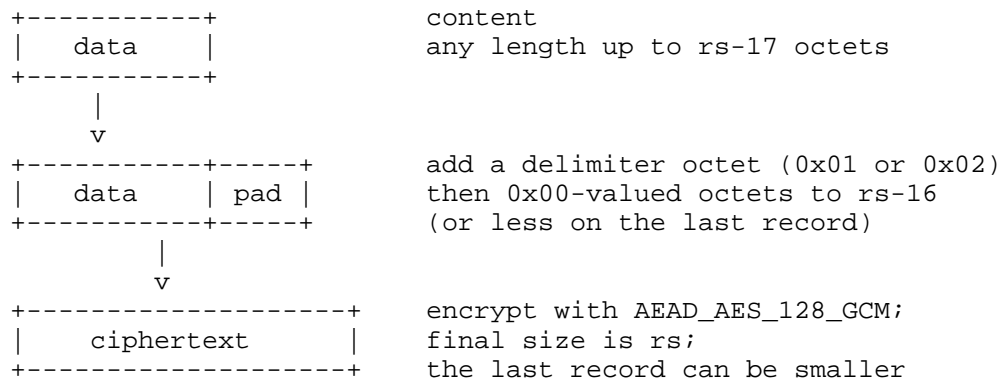
The "aes128gcm" HTTP content coding indicates that a payload has been encrypted using Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) as identified as AEAD_AES_128_GCM in [RFC5116], Section 5.1. The AEAD_AES_128_GCM algorithm uses a 128 bit content encryption key.

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 (16) | rs (4) | idlen (1) | keyid (idlen) |
+-----+-----+-----+-----+
```

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:

$$\text{PRK} = \text{HMAC-SHA-256}(\text{salt}, \text{IKM})$$

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

$$\text{cek_info} = \text{"Content-Encoding: aes128gcm"} \parallel 0x00$$

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:

$$\text{CEK} = \text{HMAC-SHA-256}(\text{PRK}, \text{cek_info} \parallel 0x01)$$

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 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_gnvG0qdGYovxyjuqRyJfJEDyoF
1Fvkj6hQPdPHI51OEUEpgz3SsLWIqS_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.

- o Name: aes128gcm
- o Description: AES-GCM encryption with a 128-bit content encryption key
- o 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:

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

```
eyJYXnIjogImRpciIsICJlbmMiOiAiQTEyOEdDTSIgfQ..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.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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:

- o The client has requested the well-known URI from the origin over an authenticated connection and a 200 (OK) response was provided, and
- o That response is fresh [RFC7234] (potentially through revalidation [RFC7232]), and
- o That response has the media type "application/json", and
- o That response's payload, when parsed as JSON [RFC7159], contains an array as the root, and
- o 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]:

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

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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:

- o 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.
- o 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.
- o 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:

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

- o restrict all numbers to be non-negative integers without fractions, and
- o 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
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.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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.

```
+-----+-----+
|           Origin-Entry (*)           ...
+-----+-----+
```

An Origin-Entry is a length-delimited string:

```
+-----+-----+
|           Origin-Len (16)           | ASCII-Origin?           ...
+-----+-----+
```

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:

- o Scheme: "https"
- o 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)
- o 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.

- o Frame Type: ORIGIN
- o Code: 0xc
- o 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

i»¿

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

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, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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:

- o 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).
- o 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.
- o 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:

- o Parameter Continuations aren't needed (Section 3.1),
- o Character Encoding and Language Information are useful, therefore a simple subset is specified (Section 3.2), and
- o 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]:

- o The document title was changed to "Indicating Character Encoding and Language for HTTP Header Field Parameters".
- o The introduction was rewritten to better explain the issues around non-ASCII characters in field values.
- o The requirement to support the "ISO-8859-1" encoding was removed.
- o 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).

- o 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:

- o "Authentication-Control", defined in [RFC8053],
- o "Authorization" (as used in HTTP Digest Authentication, defined in [RFC7616]),
- o "Content-Disposition", defined in [RFC6266], and
- o "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 Dürst 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
Münster, NW 48155
Germany

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

HTTP
Internet-Draft
Obsoletes: 6265 (if approved)
Intended status: Standards Track
Expires: 26 October 2022

L. Chen, Ed.
Google LLC
S. Englehardt, Ed.
Mozilla
M. West, Ed.
Google LLC
J. Wilander, Ed.
Apple, Inc
24 April 2022

Cookies: HTTP State Management Mechanism
draft-ietf-httpbis-rfc6265bis-10

Abstract

This document defines the HTTP Cookie and Set-Cookie header fields. These header fields can be used by HTTP servers to store state (called cookies) at HTTP user agents, letting the servers maintain a stateful session over the mostly stateless HTTP protocol. Although cookies have many historical infelicities that degrade their security and privacy, the Cookie and Set-Cookie header fields are widely used on the Internet. This document obsoletes RFC 6265.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at <https://datatracker.ietf.org/doc/draft-ietf-httpbis-rfc6265bis/>.

Discussion of this document takes place on the HTTP Working Group mailing list (<mailto: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 <https://httpwg.org/>.

Source for this draft and an issue tracker can be found at <https://github.com/httpwg/http-extensions/labels/6265bis>.

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 26 October 2022.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction	4
2. Conventions	5
2.1. Conformance Criteria	5
2.2. Syntax Notation	6
2.3. Terminology	6
3. Overview	7
3.1. Examples	8
4. Server Requirements	9
4.1. Set-Cookie	9
4.1.1. Syntax	10
4.1.2. Semantics (Non-Normative)	11
4.1.3. Cookie Name Prefixes	15
4.2. Cookie	16

4.2.1.	Syntax	16
4.2.2.	Semantics	16
5.	User Agent Requirements	17
5.1.	Subcomponent Algorithms	17
5.1.1.	Dates	17
5.1.2.	Canonicalized Host Names	19
5.1.3.	Domain Matching	20
5.1.4.	Paths and Path-Match	20
5.2.	"Same-site" and "cross-site" Requests	21
5.2.1.	Document-based requests	21
5.2.2.	Worker-based requests	22
5.3.	Ignoring Set-Cookie Header Fields	23
5.4.	The Set-Cookie Header Field	24
5.4.1.	The Expires Attribute	26
5.4.2.	The Max-Age Attribute	27
5.4.3.	The Domain Attribute	27
5.4.4.	The Path Attribute	28
5.4.5.	The Secure Attribute	28
5.4.6.	The HttpOnly Attribute	28
5.4.7.	The SameSite Attribute	28
5.5.	Storage Model	30
5.6.	Retrieval Model	36
5.6.1.	The Cookie Header Field	36
5.6.2.	Non-HTTP APIs	36
5.6.3.	Retrieval Algorithm	37
6.	Implementation Considerations	39
6.1.	Limits	39
6.2.	Application Programming Interfaces	39
6.3.	IDNA Dependency and Migration	40
7.	Privacy Considerations	40
7.1.	Third-Party Cookies	41
7.2.	Cookie Policy	41
7.3.	User Controls	42
7.4.	Expiration Dates	42
8.	Security Considerations	43
8.1.	Overview	43
8.2.	Ambient Authority	43
8.3.	Clear Text	44
8.4.	Session Identifiers	44
8.5.	Weak Confidentiality	45
8.6.	Weak Integrity	46
8.7.	Reliance on DNS	47
8.8.	SameSite Cookies	47
8.8.1.	Defense in depth	47
8.8.2.	Top-level Navigations	47
8.8.3.	Mashups and Widgets	48
8.8.4.	Server-controlled	48
8.8.5.	Reload navigations	48

8.8.6. Top-level requests with "unsafe" methods	49
9. IANA Considerations	50
9.1. Cookie	50
9.2. Set-Cookie	50
9.3. Cookie Attribute Registry	50
9.3.1. Procedure	51
9.3.2. Registration	51
10. References	51
10.1. Normative References	51
10.2. Informative References	53
Appendix A. Changes	55
A.1. draft-ietf-httpbis-rfc6265bis-00	55
A.2. draft-ietf-httpbis-rfc6265bis-01	55
A.3. draft-ietf-httpbis-rfc6265bis-02	56
A.4. draft-ietf-httpbis-rfc6265bis-03	56
A.5. draft-ietf-httpbis-rfc6265bis-04	57
A.6. draft-ietf-httpbis-rfc6265bis-05	57
A.7. draft-ietf-httpbis-rfc6265bis-06	57
A.8. draft-ietf-httpbis-rfc6265bis-07	58
A.9. draft-ietf-httpbis-rfc6265bis-08	58
A.10. draft-ietf-httpbis-rfc6265bis-09	59
A.11. draft-ietf-httpbis-rfc6265bis-10	59
Acknowledgements	60
Authors' Addresses	60

1. Introduction

This document defines the HTTP Cookie and Set-Cookie header fields. Using the Set-Cookie header field, an HTTP server can pass name/value pairs and associated metadata (called cookies) to a user agent. When the user agent makes subsequent requests to the server, the user agent uses the metadata and other information to determine whether to return the name/value pairs in the Cookie header field.

Although simple on their surface, cookies have a number of complexities. For example, the server indicates a scope for each cookie when sending it to the user agent. The scope indicates the maximum amount of time in which the user agent should return the cookie, the servers to which the user agent should return the cookie, and the URI schemes for which the cookie is applicable.

For historical reasons, cookies contain a number of security and privacy infelicities. For example, a server can indicate that a given cookie is intended for "secure" connections, but the Secure attribute does not provide integrity in the presence of an active network attacker. Similarly, cookies for a given host are shared across all the ports on that host, even though the usual "same-origin policy" used by web browsers isolates content retrieved via different ports.

There are two audiences for this specification: developers of cookie-generating servers and developers of cookie-consuming user agents.

To maximize interoperability with user agents, servers SHOULD limit themselves to the well-behaved profile defined in Section 4 when generating cookies.

User agents MUST implement the more liberal processing rules defined in Section 5, in order to maximize interoperability with existing servers that do not conform to the well-behaved profile defined in Section 4.

This document specifies the syntax and semantics of these header fields as they are actually used on the Internet. In particular, this document does not create new syntax or semantics beyond those in use today. The recommendations for cookie generation provided in Section 4 represent a preferred subset of current server behavior, and even the more liberal cookie processing algorithm provided in Section 5 does not recommend all of the syntactic and semantic variations in use today. Where some existing software differs from the recommended protocol in significant ways, the document contains a note explaining the difference.

This document obsoletes [RFC6265].

2. Conventions

2.1. Conformance Criteria

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].

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("MUST", "SHOULD", "MAY", etc.) used in introducing the algorithm.

Conformance requirements phrased as algorithms or specific steps can be implemented in any manner, so long as the end result is equivalent. In particular, the algorithms defined in this specification are intended to be easy to understand and are not intended to be performant.

2.2. Syntax Notation

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

The following core rules are included by reference, as defined in [RFC5234], Appendix B.1: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTLs (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), NUL (null octet), OCTET (any 8-bit sequence of data except NUL), SP (space), HTAB (horizontal tab), CHAR (any [USASCII] character), VCHAR (any visible [USASCII] character), and WSP (whitespace).

The OWS (optional whitespace) and BWS (bad whitespace) rules are defined in Section 5.6.3 of [HTTPSEM].

2.3. Terminology

The terms "user agent", "client", "server", "proxy", and "origin server" have the same meaning as in the HTTP/1.1 specification ([HTTPSEM], Section 3).

The request-host is the name of the host, as known by the user agent, to which the user agent is sending an HTTP request or from which it is receiving an HTTP response (i.e., the name of the host to which it sent the corresponding HTTP request).

The term request-uri refers to "target URI" as defined in Section 7.1 of [HTTPSEM].

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 term string means a sequence of non-NUL octets.

The terms "active document", "ancestor browsing context", "browsing context", "dedicated worker", "Document", "nested browsing context", "opaque origin", "parent browsing context", "sandboxed origin browsing context flag", "shared worker", "the worker's Documents", "top-level browsing context", and "WorkerGlobalScope" 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 9.2.1 of [HTTPSEM].

A domain's "public suffix" is the portion of a domain that is controlled by a public registry, such as "com", "co.uk", and "pvt.k12.wy.us". A domain's "registrable domain" is the domain's public suffix plus the label to its left. That is, for `https://www.site.example`, the public suffix is `example`, and the registrable domain is `site.example`. Whenever possible, user agents SHOULD use an up-to-date public suffix list, such as the one maintained by the Mozilla project at [PSL].

The term "request", as well as a request's "client", "current url", "method", "target browsing context", and "url list", are defined in [FETCH].

The term "non-HTTP APIs" refers to non-HTTP mechanisms used to set and retrieve cookies, such as a web browser API that exposes cookies to scripts.

3. Overview

This section outlines a way for an origin server to send state information to a user agent and for the user agent to return the state information to the origin server.

To store state, the origin server includes a Set-Cookie header field in an HTTP response. In subsequent requests, the user agent returns a Cookie request header field to the origin server. The Cookie header field contains cookies the user agent received in previous Set-Cookie header fields. The origin server is free to ignore the Cookie header field or use its contents for an application-defined purpose.

Origin servers MAY send a Set-Cookie response header field with any response. An origin server can include multiple Set-Cookie header fields in a single response. The presence of a Cookie or a Set-Cookie header field does not preclude HTTP caches from storing and reusing a response.

Origin servers SHOULD NOT fold multiple Set-Cookie header fields into a single header field. The usual mechanism for folding HTTP headers fields (i.e., as defined in Section 5.3 of [HTTPSEM]) might change the semantics of the Set-Cookie header field because the %x2C (",") character is used by Set-Cookie in a way that conflicts with such folding.

User agents MAY ignore Set-Cookie header fields based on response status codes or the user agent's cookie policy (see Section 5.3).

3.1. Examples

Using the Set-Cookie header field, a server can send the user agent a short string in an HTTP response that the user agent will return in future HTTP requests that are within the scope of the cookie. For example, the server can send the user agent a "session identifier" named SID with the value 31d4d96e407aad42. The user agent then returns the session identifier in subsequent requests.

```
== Server -> User Agent ==
```

```
Set-Cookie: SID=31d4d96e407aad42
```

```
== User Agent -> Server ==
```

```
Cookie: SID=31d4d96e407aad42
```

The server can alter the default scope of the cookie using the Path and Domain attributes. For example, the server can instruct the user agent to return the cookie to every path and every subdomain of site.example.

```
== Server -> User Agent ==
```

```
Set-Cookie: SID=31d4d96e407aad42; Path=/; Domain=site.example
```

```
== User Agent -> Server ==
```

```
Cookie: SID=31d4d96e407aad42
```

As shown in the next example, the server can store multiple cookies at the user agent. For example, the server can store a session identifier as well as the user's preferred language by returning two Set-Cookie header fields. Notice that the server uses the Secure and HttpOnly attributes to provide additional security protections for the more sensitive session identifier (see Section 4.1.2).

== Server -> User Agent ==

```
Set-Cookie: SID=31d4d96e407aad42; Path=/; Secure; HttpOnly
Set-Cookie: lang=en-US; Path=/; Domain=site.example
```

== User Agent -> Server ==

```
Cookie: SID=31d4d96e407aad42; lang=en-US
```

Notice that the Cookie header field above contains two cookies, one named SID and one named lang. If the server wishes the user agent to persist the cookie over multiple "sessions" (e.g., user agent restarts), the server can specify an expiration date in the Expires attribute. Note that the user agent might delete the cookie before the expiration date if the user agent's cookie store exceeds its quota or if the user manually deletes the server's cookie.

== Server -> User Agent ==

```
Set-Cookie: lang=en-US; Expires=Wed, 09 Jun 2021 10:18:14 GMT
```

== User Agent -> Server ==

```
Cookie: SID=31d4d96e407aad42; lang=en-US
```

Finally, to remove a cookie, the server returns a Set-Cookie header field with an expiration date in the past. The server will be successful in removing the cookie only if the Path and the Domain attribute in the Set-Cookie header field match the values used when the cookie was created.

== Server -> User Agent ==

```
Set-Cookie: lang=; Expires=Sun, 06 Nov 1994 08:49:37 GMT
```

== User Agent -> Server ==

```
Cookie: SID=31d4d96e407aad42
```

4. Server Requirements

This section describes the syntax and semantics of a well-behaved profile of the Cookie and Set-Cookie header fields.

4.1. Set-Cookie

The Set-Cookie HTTP response header field is used to send cookies from the server to the user agent.

4.1.1. Syntax

Informally, the Set-Cookie response header field contains a cookie, which begins with a name-value-pair, followed by zero or more attribute-value pairs. Servers SHOULD NOT send Set-Cookie header fields that fail to conform to the following grammar:

```

set-cookie           = set-cookie-string
set-cookie-string    = BWS cookie-pair *( BWS ";" OWS cookie-av )
cookie-pair          = cookie-name BWS "=" BWS cookie-value
cookie-name          = 1*cookie-octet
cookie-value         = *cookie-octet / ( DQUOTE *cookie-octet DQUOTE )
cookie-octet         = %x21 / %x23-2B / %x2D-3A / %x3C-5B / %x5D-7E
                      / %x80-FF
                      ; octets excluding CTLs,
                      ; whitespace DQUOTE, comma, semicolon,
                      ; and backslash

cookie-av            = expires-av / max-age-av / domain-av /
                      path-av / secure-av / httponly-av /
                      samesite-av / extension-av
expires-av           = "Expires" BWS "=" BWS sane-cookie-date
sane-cookie-date     =
    <IMF-fixdate, defined in [HTTPSEM], Section 5.6.7>
max-age-av           = "Max-Age" BWS "=" BWS non-zero-digit *DIGIT
non-zero-digit       = %x31-39
                      ; digits 1 through 9
domain-av            = "Domain" BWS "=" BWS domain-value
domain-value         = <subdomain>
                      ; see details below
path-av              = "Path" BWS "=" BWS path-value
path-value           = *av-octet
secure-av            = "Secure"
httponly-av          = "HttpOnly"
samesite-av          = "SameSite" BWS "=" BWS samesite-value
samesite-value       = "Strict" / "Lax" / "None"
extension-av         = *av-octet
av-octet             = %x20-3A / %x3C-7E
                      ; any CHAR except CTLs or ";"

```

Note that some of the grammatical terms above reference documents that use different grammatical notations than this document (which uses ABNF from [RFC5234]).

The semantics of the cookie-value are not defined by this document.

To maximize compatibility with user agents, servers that wish to store arbitrary data in a cookie-value SHOULD encode that data, for example, using Base64 [RFC4648].

The domain-value is a subdomain as defined by [RFC1034], Section 3.5, and as enhanced by [RFC1123], Section 2.1. Thus, domain-value is a string of [USASCII] characters, such as one obtained by applying the "ToASCII" operation defined in Section 4 of [RFC3490].

Per the grammar above, the cookie-value MAY be wrapped in DQUOTE characters. Note that in this case, the initial and trailing DQUOTE characters are not stripped. They are part of the cookie-value, and will be included in Cookie header fields sent to the server.

The portions of the set-cookie-string produced by the cookie-av term are known as attributes. To maximize compatibility with user agents, servers SHOULD NOT produce two attributes with the same name in the same set-cookie-string. (See Section 5.5 for how user agents handle this case.)

Servers SHOULD NOT include more than one Set-Cookie header field in the same response with the same cookie-name. (See Section 5.4 for how user agents handle this case.)

If a server sends multiple responses containing Set-Cookie header fields concurrently to the user agent (e.g., when communicating with the user agent over multiple sockets), these responses create a "race condition" that can lead to unpredictable behavior.

NOTE: Some existing user agents differ in their interpretation of two-digit years. To avoid compatibility issues, servers SHOULD use the rfc1123-date format, which requires a four-digit year.

NOTE: Some user agents store and process dates in cookies as 32-bit UNIX time_t values. Implementation bugs in the libraries supporting time_t processing on some systems might cause such user agents to process dates after the year 2038 incorrectly.

4.1.2. Semantics (Non-Normative)

This section describes simplified semantics of the Set-Cookie header field. These semantics are detailed enough to be useful for understanding the most common uses of cookies by servers. The full semantics are described in Section 5.

When the user agent receives a Set-Cookie header field, the user agent stores the cookie together with its attributes. Subsequently, when the user agent makes an HTTP request, the user agent includes the applicable, non-expired cookies in the Cookie header field.

If the user agent receives a new cookie with the same cookie-name, domain-value, and path-value as a cookie that it has already stored, the existing cookie is evicted and replaced with the new cookie. Notice that servers can delete cookies by sending the user agent a new cookie with an Expires attribute with a value in the past.

Unless the cookie's attributes indicate otherwise, the cookie is returned only to the origin server (and not, for example, to any subdomains), and it expires at the end of the current session (as defined by the user agent). User agents ignore unrecognized cookie attributes (but not the entire cookie).

4.1.2.1. The Expires Attribute

The Expires attribute indicates the maximum lifetime of the cookie, represented as the date and time at which the cookie expires. The user agent is not required to retain the cookie until the specified date has passed. In fact, user agents often evict cookies due to memory pressure or privacy concerns.

The user agent MUST limit the maximum value of the Expires attribute. The limit SHOULD NOT be greater than 400 days (34560000 seconds) in the future. The RECOMMENDED limit is 400 days in the future, but the user agent MAY adjust the limit (see Section 7.2). Expires attributes that are greater than the limit MUST be reduced to the limit.

4.1.2.2. The Max-Age Attribute

The Max-Age attribute indicates the maximum lifetime of the cookie, represented as the number of seconds until the cookie expires. The user agent is not required to retain the cookie for the specified duration. In fact, user agents often evict cookies due to memory pressure or privacy concerns.

The user agent MUST limit the maximum value of the Max-Age attribute. The limit SHOULD NOT be greater than 400 days (34560000 seconds) in duration. The RECOMMENDED limit is 400 days in duration, but the user agent MAY adjust the limit (see Section 7.2). Max-Age attributes that are greater than the limit MUST be reduced to the limit.

NOTE: Some existing user agents do not support the Max-Age attribute. User agents that do not support the Max-Age attribute ignore the attribute.

If a cookie has both the Max-Age and the Expires attribute, the Max-Age attribute has precedence and controls the expiration date of the cookie. If a cookie has neither the Max-Age nor the Expires attribute, the user agent will retain the cookie until "the current session is over" (as defined by the user agent).

4.1.2.3. The Domain Attribute

The Domain attribute specifies those hosts to which the cookie will be sent. For example, if the value of the Domain attribute is "site.example", the user agent will include the cookie in the Cookie header field when making HTTP requests to site.example, www.site.example, and www.corp.site.example. (Note that a leading %x2E ("."), if present, is ignored even though that character is not permitted, but a trailing %x2E ("."), if present, will cause the user agent to ignore the attribute.) If the server omits the Domain attribute, the user agent will return the cookie only to the origin server.

WARNING: Some existing user agents treat an absent Domain attribute as if the Domain attribute were present and contained the current host name. For example, if site.example returns a Set-Cookie header field without a Domain attribute, these user agents will erroneously send the cookie to www.site.example as well.

The user agent will reject cookies unless the Domain attribute specifies a scope for the cookie that would include the origin server. For example, the user agent will accept a cookie with a Domain attribute of "site.example" or of "foo.site.example" from foo.site.example, but the user agent will not accept a cookie with a Domain attribute of "bar.site.example" or of "baz.foo.site.example".

NOTE: For security reasons, many user agents are configured to reject Domain attributes that correspond to "public suffixes". For example, some user agents will reject Domain attributes of "com" or "co.uk". (See Section 5.5 for more information.)

4.1.2.4. The Path Attribute

The scope of each cookie is limited to a set of paths, controlled by the Path attribute. If the server omits the Path attribute, the user agent will use the "directory" of the request-uri's path component as the default value. (See Section 5.1.4 for more details.)

The user agent will include the cookie in an HTTP request only if the path portion of the request-uri matches (or is a subdirectory of) the cookie's Path attribute, where the %x2F ("/") character is interpreted as a directory separator.

Although seemingly useful for isolating cookies between different paths within a given host, the Path attribute cannot be relied upon for security (see Section 8).

4.1.2.5. The Secure Attribute

The Secure attribute limits the scope of the cookie to "secure" channels (where "secure" is defined by the user agent). When a cookie has the Secure attribute, the user agent will include the cookie in an HTTP request only if the request is transmitted over a secure channel (typically HTTP over Transport Layer Security (TLS) [RFC2818]).

4.1.2.6. The HttpOnly Attribute

The HttpOnly attribute limits the scope of the cookie to HTTP requests. In particular, the attribute instructs the user agent to omit the cookie when providing access to cookies via non-HTTP APIs.

Note that the HttpOnly attribute is independent of the Secure attribute: a cookie can have both the HttpOnly and the Secure attribute.

4.1.2.7. The SameSite Attribute

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 5.2. For example, requests for `https://site.example/sekrit-image` will attach same-site cookies if and only if initiated from a context whose "site for cookies" is an origin with a scheme and registered domain of "https" and "site.example" respectively.

If the "SameSite" attribute's value is "Strict", 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 5.4.7.1. If the value is "None", the cookie will be sent with same-site and cross-site requests. If the "SameSite" attribute's value is something other than these three known keywords, the attribute's value will be subject to a default enforcement mode that is equivalent to "Lax".

The "SameSite" attribute affects cookie creation as well as delivery. Cookies which assert "SameSite=Lax" or "SameSite=Strict" cannot be set in responses to cross-site subresource requests, or cross-site nested navigations. They can be set along with any top-level navigation, cross-site or otherwise.

4.1.3. Cookie Name Prefixes

Section 8.5 and Section 8.6 of this document spell out some of the drawbacks of cookies' historical implementation. In particular, it is impossible for a server to have confidence that a given cookie was set with a particular set of attributes. In order to provide such confidence in a backwards-compatible way, two common sets of requirements can be inferred from the first few characters of the cookie's name.

The normative requirements for the prefixes described below are detailed in the storage model algorithm defined in Section 5.5.

4.1.3.1. The "__Secure-" Prefix

If a cookie's name begins with a case-sensitive match for the string `__Secure-`, then the cookie will have been set with a Secure attribute.

For example, the following Set-Cookie header field would be rejected by a conformant user agent, as it does not have a Secure attribute.

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

Whereas the following Set-Cookie header field would be accepted if set from a secure origin (e.g. "https://site.example/"), and rejected otherwise:

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

4.1.3.2. The "__Host-" Prefix

If a cookie's name begins with a case-sensitive match for the string `__Host-`, then the cookie will have been set with a Secure attribute, a Path attribute with a value of `/`, and no Domain attribute.

This combination yields a cookie that hews as closely as a cookie can to treating the origin as a security boundary. The lack of a Domain attribute ensures that the cookie's host-only-flag is true, locking the cookie to a particular host, rather than allowing it to span subdomains. Setting the Path to / means that the cookie is effective for the entire host, and won't be overridden for specific paths. The Secure attribute ensures that the cookie is unaltered by non-secure origins, and won't span protocols.

Ports are the only piece of the origin model that __Host- cookies continue to ignore.

For example, the following cookies would always be rejected:

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

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

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

4.2. Cookie

4.2.1. Syntax

The user agent sends stored cookies to the origin server in the Cookie header field. If the server conforms to the requirements in Section 4.1 (and the user agent conforms to the requirements in Section 5), the user agent will send a Cookie header field that conforms to the following grammar:

```
cookie          = cookie-string
cookie-string   = cookie-pair *( ";" SP cookie-pair )
```

4.2.2. Semantics

Each cookie-pair represents a cookie stored by the user agent. The cookie-pair contains the cookie-name and cookie-value the user agent received in the Set-Cookie header field.

Notice that the cookie attributes are not returned. In particular, the server cannot determine from the Cookie field alone when a cookie will expire, for which hosts the cookie is valid, for which paths the cookie is valid, or whether the cookie was set with the Secure or HttpOnly attributes.

The semantics of individual cookies in the Cookie header field are not defined by this document. Servers are expected to imbue these cookies with application-specific semantics.

Although cookies are serialized linearly in the Cookie header field, servers SHOULD NOT rely upon the serialization order. In particular, if the Cookie header field contains two cookies with the same name (e.g., that were set with different Path or Domain attributes), servers SHOULD NOT rely upon the order in which these cookies appear in the header field.

5. User Agent Requirements

This section specifies the Cookie and Set-Cookie header fields in sufficient detail that a user agent implementing these requirements precisely can interoperate with existing servers (even those that do not conform to the well-behaved profile described in Section 4).

A user agent could enforce more restrictions than those specified herein (e.g., restrictions specified by its cookie policy, described in Section 7.2). However, such additional restrictions may reduce the likelihood that a user agent will be able to interoperate with existing servers.

5.1. Subcomponent Algorithms

This section defines some algorithms used by user agents to process specific subcomponents of the Cookie and Set-Cookie header fields.

5.1.1. Dates

The user agent MUST use an algorithm equivalent to the following algorithm to parse a cookie-date. Note that the various boolean flags defined as a part of the algorithm (i.e., found-time, found-day-of-month, found-month, found-year) are initially "not set".

1. Using the grammar below, divide the cookie-date into date-tokens.

```

cookie-date      = *delimiter date-token-list *delimiter
date-token-list = date-token *( 1*delimiter date-token )
date-token       = 1*non-delimiter

delimiter        = %x09 / %x20-2F / %x3B-40 / %x5B-60 / %x7B-7E
non-delimiter     = %x00-08 / %x0A-1F / DIGIT / ":" / ALPHA
                  / %x7F-FF
non-digit         = %x00-2F / %x3A-FF

day-of-month      = 1*2DIGIT [ non-digit *OCTET ]
month             = ( "jan" / "feb" / "mar" / "apr" /
                    "may" / "jun" / "jul" / "aug" /
                    "sep" / "oct" / "nov" / "dec" ) *OCTET
year              = 2*4DIGIT [ non-digit *OCTET ]
time              = hms-time [ non-digit *OCTET ]
hms-time          = time-field ":" time-field ":" time-field
time-field        = 1*2DIGIT

```

2. Process each date-token sequentially in the order the date-tokens appear in the cookie-date:
 1. If the found-time flag is not set and the token matches the time production, set the found-time flag and set the hour-value, minute-value, and second-value to the numbers denoted by the digits in the date-token, respectively. Skip the remaining sub-steps and continue to the next date-token.
 2. If the found-day-of-month flag is not set and the date-token matches the day-of-month production, set the found-day-of-month flag and set the day-of-month-value to the number denoted by the date-token. Skip the remaining sub-steps and continue to the next date-token.
 3. If the found-month flag is not set and the date-token matches the month production, set the found-month flag and set the month-value to the month denoted by the date-token. Skip the remaining sub-steps and continue to the next date-token.
 4. If the found-year flag is not set and the date-token matches the year production, set the found-year flag and set the year-value to the number denoted by the date-token. Skip the remaining sub-steps and continue to the next date-token.
3. If the year-value is greater than or equal to 70 and less than or equal to 99, increment the year-value by 1900.
4. If the year-value is greater than or equal to 0 and less than or equal to 69, increment the year-value by 2000.

1. NOTE: Some existing user agents interpret two-digit years differently.
5. Abort these steps and fail to parse the cookie-date if:
 - * at least one of the found-day-of-month, found-month, found-year, or found-time flags is not set,
 - * the day-of-month-value is less than 1 or greater than 31,
 - * the year-value is less than 1601,
 - * the hour-value is greater than 23,
 - * the minute-value is greater than 59, or
 - * the second-value is greater than 59.

(Note that leap seconds cannot be represented in this syntax.)
6. Let the parsed-cookie-date be the date whose day-of-month, month, year, hour, minute, and second (in UTC) are the day-of-month-value, the month-value, the year-value, the hour-value, the minute-value, and the second-value, respectively. If no such date exists, abort these steps and fail to parse the cookie-date.
7. Return the parsed-cookie-date as the result of this algorithm.

5.1.2. Canonicalized Host Names

A canonicalized host name is the string generated by the following algorithm:

1. Convert the host name to a sequence of individual domain name labels.
2. Convert each label that is not a Non-Reserved LDH (NR-LDH) label, to an A-label (see Section 2.3.2.1 of [RFC5890] for the former and latter), or to a "punycode label" (a label resulting from the "ToASCII" conversion in Section 4 of [RFC3490]), as appropriate (see Section 6.3 of this specification).
3. Concatenate the resulting labels, separated by a %x2E (".") character.

5.1.3. Domain Matching

A string domain-matches a given domain string if at least one of the following conditions hold:

- * The domain string and the string are identical. (Note that both the domain string and the string will have been canonicalized to lower case at this point.)
- * All of the following conditions hold:
 - The domain string is a suffix of the string.
 - The last character of the string that is not included in the domain string is a %x2E (".") character.
 - The string is a host name (i.e., not an IP address).

5.1.4. Paths and Path-Match

The user agent MUST use an algorithm equivalent to the following algorithm to compute the default-path of a cookie:

1. Let uri-path be the path portion of the request-uri if such a portion exists (and empty otherwise).
2. If the uri-path is empty or if the first character of the uri-path is not a %x2F ("/") character, output %x2F ("/") and skip the remaining steps.
3. If the uri-path contains no more than one %x2F ("/") character, output %x2F ("/") and skip the remaining step.
4. Output the characters of the uri-path from the first character up to, but not including, the right-most %x2F ("/").

A request-path path-matches a given cookie-path if at least one of the following conditions holds:

- * The cookie-path and the request-path are identical.

Note that this differs from the rules in [RFC3986] for equivalence of the path component, and hence two equivalent paths can have different cookies.

- * The cookie-path is a prefix of the request-path, and the last character of the cookie-path is %x2F ("/").

- * The cookie-path is a prefix of the request-path, and the first character of the request-path that is not included in the cookie-path is a %x2F ("/") character.

5.2. "Same-site" and "cross-site" Requests

Two origins are same-site if they satisfy the "same site" criteria defined in [SAMESITE]. A request is "same-site" if the following criteria are true:

1. The request is not the result of a cross-site redirect. That is, the origin of every url in the request's url list is same-site with the request's current url's origin.
2. The request is not the result of a reload navigation triggered through a user interface element (as defined by the user agent; e.g., a request triggered by the user clicking a refresh button on a toolbar).
3. The request's current url's origin is same-site with the request's client's "site for cookies" (which is an origin), or if the request has no client or the request's client is null.

Requests which are the result of a reload navigation triggered through a user interface element are same-site if the reloaded document was originally navigated to via a same-site request. A request that is not "same-site" is instead "cross-site".

The request's client's "site for cookies" is calculated depending upon its client's type, as described in the following subsections:

5.2.1. 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 origin of that URI represents the context in which a user most likely believes themselves to be interacting. We'll define this origin, the top-level browsing context's active document's origin, as the "top-level origin".

For a document displayed in a top-level browsing context, we can stop here: the document's "site for cookies" is the top-level origin.

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]. A document's

"site for cookies" is the top-level origin if and only if the top-level origin is same-site with the document's origin, and with each of the document's ancestor documents' origins. Otherwise its "site for cookies" is an origin set to an opaque origin.

Given a Document (document), the following algorithm returns its "site for cookies":

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 is not same-site with top-origin, return an origin set to an opaque origin.
5. Return top-origin.

5.2.2. 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.

5.2.2.1. 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 cookies" values, the worker's "site for cookies" will be an origin set to an opaque origin in cases where the values are not all same-site with the worker's origin, and the worker's origin in cases where the values agree.

Given a `WorkerGlobalScope` (`worker`), the following algorithm returns its "site for cookies":

1. Let `site` be `worker`'s origin.
2. For each document in `worker`'s Documents:
 1. Let `document-site` be document's "site for cookies" (as defined in Section 5.2.1).
 2. If `document-site` is not same-site with `site`, return an origin set to an opaque origin.
3. Return `site`.

5.2.2.2. 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 5.2.1 and Section 5.2.2.1

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 Service Worker's URI's origin.

Given a `ServiceWorkerGlobalScope` (`worker`), the following algorithm returns its "site for cookies":

1. Return `worker`'s origin.

5.3. Ignoring Set-Cookie Header Fields

User agents MAY ignore Set-Cookie header fields contained in responses with 100-level status codes or based on its cookie policy (see Section 7.2).

All other Set-Cookie header fields SHOULD be processed according to Section 5.4. That is, Set-Cookie header fields contained in responses with non-100-level status codes (including those in responses with 400- and 500-level status codes) SHOULD be processed unless ignored according to the user agent's cookie policy.

5.4. The Set-Cookie Header Field

When a user agent receives a Set-Cookie header field in an HTTP response, the user agent MAY ignore the Set-Cookie header field in its entirety (see Section 5.3).

If the user agent does not ignore the Set-Cookie header field in its entirety, the user agent MUST parse the field-value of the Set-Cookie header field as a set-cookie-string (defined below).

NOTE: The algorithm below is more permissive than the grammar in Section 4.1. For example, the algorithm strips leading and trailing whitespace from the cookie name and value (but maintains internal whitespace), whereas the grammar in Section 4.1 forbids whitespace in these positions. In addition, the algorithm below accommodates some characters that are not cookie-octets according to the grammar in Section 4.1. User agents use this algorithm so as to interoperate with servers that do not follow the recommendations in Section 4.

NOTE: As set-cookie-string may originate from a non-HTTP API, it is not guaranteed to be free of CTL characters, so this algorithm handles them explicitly. Horizontal tab (%x09) is excluded from the CTL characters that lead to set-cookie-string rejection, as it is considered whitespace, which is handled separately.

NOTE: The set-cookie-string may contain octet sequences that appear percent-encoded as per Section 2.1 of [RFC3986]. However, a user agent MUST NOT decode these sequences and instead parse the individual octets as specified in this algorithm.

A user agent MUST use an algorithm equivalent to the following algorithm to parse a set-cookie-string:

1. If the set-cookie-string contains a %x00-08 / %x0A-1F / %x7F character (CTL characters excluding HTAB): Abort these steps and ignore the set-cookie-string entirely.
2. If the set-cookie-string contains a %x3B (";") character:

1. The name-value-pair string consists of the characters up to, but not including, the first %x3B (";"), and the unparsed-attributes consist of the remainder of the set-cookie-string (including the %x3B (";") in question).

Otherwise:

1. The name-value-pair string consists of all the characters contained in the set-cookie-string, and the unparsed-attributes is the empty string.
3. If the name-value-pair string lacks a %x3D ("=") character, then the name string is empty, and the value string is the value of name-value-pair.

Otherwise, the name string consists of the characters up to, but not including, the first %x3D ("=") character, and the (possibly empty) value string consists of the characters after the first %x3D ("=") character.

4. Remove any leading or trailing WSP characters from the name string and the value string.
5. If the sum of the lengths of the name string and the value string is more than 4096 octets, abort these steps and ignore the set-cookie-string entirely.
6. The cookie-name is the name string, and the cookie-value is the value string.

The user agent MUST use an algorithm equivalent to the following algorithm to parse the unparsed-attributes:

1. If the unparsed-attributes string is empty, skip the rest of these steps.
2. Discard the first character of the unparsed-attributes (which will be a %x3B (";") character).
3. If the remaining unparsed-attributes contains a %x3B (";") character:
 1. Consume the characters of the unparsed-attributes up to, but not including, the first %x3B (";") character.

Otherwise:

1. Consume the remainder of the unparsed-attributes.

Let the cookie-av string be the characters consumed in this step.

4. If the cookie-av string contains a %x3D ("=") character:
 1. The (possibly empty) attribute-name string consists of the characters up to, but not including, the first %x3D ("=") character, and the (possibly empty) attribute-value string consists of the characters after the first %x3D ("=") character.
- Otherwise:
 1. The attribute-name string consists of the entire cookie-av string, and the attribute-value string is empty.
5. Remove any leading or trailing WSP characters from the attribute-name string and the attribute-value string.
6. If the attribute-value is longer than 1024 octets, ignore the cookie-av string and return to Step 1 of this algorithm.
7. Process the attribute-name and attribute-value according to the requirements in the following subsections. (Notice that attributes with unrecognized attribute-names are ignored.)
8. Return to Step 1 of this algorithm.

When the user agent finishes parsing the set-cookie-string, the user agent is said to "receive a cookie" from the request-uri with name cookie-name, value cookie-value, and attributes cookie-attribute-list. (See Section 5.5 for additional requirements triggered by receiving a cookie.)

5.4.1. The Expires Attribute

If the attribute-name case-insensitively matches the string "Expires", the user agent MUST process the cookie-av as follows.

1. Let the expiry-time be the result of parsing the attribute-value as cookie-date (see Section 5.1.1).
2. If the attribute-value failed to parse as a cookie date, ignore the cookie-av.
3. Let cookie-age-limit be the maximum age of the cookie (which SHOULD be 400 days in the future or sooner, see Section 4.1.2.1).

4. If the expiry-time is more than cookie-age-limit, the user agent MUST set the expiry time to cookie-age-limit in seconds.
5. If the expiry-time is earlier than the earliest date the user agent can represent, the user agent MAY replace the expiry-time with the earliest representable date.
6. Append an attribute to the cookie-attribute-list with an attribute-name of Expires and an attribute-value of expiry-time.

5.4.2. The Max-Age Attribute

If the attribute-name case-insensitively matches the string "Max-Age", the user agent MUST process the cookie-av as follows.

1. If the first character of the attribute-value is not a DIGIT or a "-" character, ignore the cookie-av.
2. If the remainder of attribute-value contains a non-DIGIT character, ignore the cookie-av.
3. Let delta-seconds be the attribute-value converted to an integer.
4. Let cookie-age-limit be the maximum age of the cookie (which SHOULD be 400 days or less, see Section 4.1.2.2).
5. Set delta-seconds to the smaller of its present value and cookie-age-limit.
6. If delta-seconds is less than or equal to zero (0), let expiry-time be the earliest representable date and time. Otherwise, let the expiry-time be the current date and time plus delta-seconds seconds.
7. Append an attribute to the cookie-attribute-list with an attribute-name of Max-Age and an attribute-value of expiry-time.

5.4.3. The Domain Attribute

If the attribute-name case-insensitively matches the string "Domain", the user agent MUST process the cookie-av as follows.

1. Let cookie-domain be the attribute-value.
2. If cookie-domain starts with %x2E ("."), let cookie-domain be cookie-domain without its leading %x2E (".").
3. Convert the cookie-domain to lower case.

4. Append an attribute to the cookie-attribute-list with an attribute-name of Domain and an attribute-value of cookie-domain.

5.4.4. The Path Attribute

If the attribute-name case-insensitively matches the string "Path", the user agent MUST process the cookie-av as follows.

1. If the attribute-value is empty or if the first character of the attribute-value is not %x2F ("/"):

1. Let cookie-path be the default-path.

Otherwise:

1. Let cookie-path be the attribute-value.

2. Append an attribute to the cookie-attribute-list with an attribute-name of Path and an attribute-value of cookie-path.

5.4.5. The Secure Attribute

If the attribute-name case-insensitively matches the string "Secure", the user agent MUST append an attribute to the cookie-attribute-list with an attribute-name of Secure and an empty attribute-value.

5.4.6. The HttpOnly Attribute

If the attribute-name case-insensitively matches the string "HttpOnly", the user agent MUST append an attribute to the cookie-attribute-list with an attribute-name of HttpOnly and an empty attribute-value.

5.4.7. The SameSite Attribute

If the attribute-name case-insensitively matches the string "SameSite", the user agent MUST process the cookie-av as follows:

1. Let enforcement be "Default".
2. If cookie-av's attribute-value is a case-insensitive match for "None", set enforcement to "None".
3. If cookie-av's attribute-value is a case-insensitive match for "Strict", set enforcement to "Strict".
4. If cookie-av's attribute-value is a case-insensitive match for "Lax", set enforcement to "Lax".

5. Append an attribute to the cookie-attribute-list with an attribute-name of "SameSite" and an attribute-value of enforcement.

5.4.7.1. "Strict" and "Lax" enforcement

Same-site cookies in "Strict" enforcement mode will not be sent along with top-level navigations which are triggered from a cross-site document context. As discussed in Section 8.8.2, 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 [HTTPSEM] sense) HTTP method. (Note that a request's method may be changed from POST to GET for some redirects (see Sections 15.4.2 and 15.4.3 of [HTTPSEM]); in these cases, a request's "safe"ness is determined based on the method of the current redirect hop.)

Lax enforcement provides reasonable defense in depth against CSRF attacks that rely on unsafe HTTP methods (like POST), but does 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 5.2.1), 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 8.8.2 to mitigate the risk of CSRF more completely.

5.4.7.2. "Lax-Allowing-Unsafe" enforcement

As discussed in Section 8.8.6, compatibility concerns may necessitate the use of a "Lax-allowing-unsafe" enforcement mode that allows cookies to be sent with a cross-site HTTP request if and only if it is a top-level request, regardless of request method. That is, the "Lax-allowing-unsafe" enforcement mode waives the requirement for the HTTP request's method to be "safe" in the SameSite enforcement step of the retrieval algorithm in Section 5.6.3. (All cookies, regardless of SameSite enforcement mode, may be set for top-level navigations, regardless of HTTP request method, as specified in

Section 5.5.)

"Lax-allowing-unsafe" is not a distinct value of the SameSite attribute. Rather, user agents MAY apply "Lax-allowing-unsafe" enforcement only to cookies that did not explicitly specify a SameSite attribute (i.e., those whose same-site-flag was set to "Default" by default). To limit the scope of this compatibility mode, user agents which apply "Lax-allowing-unsafe" enforcement SHOULD restrict the enforcement to cookies which were created recently. Deployment experience has shown a cookie age of 2 minutes or less to be a reasonable limit.

If the user agent uses "Lax-allowing-unsafe" enforcement, it MUST apply the following modification to the retrieval algorithm defined in Section 5.6.3:

Replace the condition in the penultimate bullet point of step 1 of the retrieval algorithm reading

- * The HTTP request associated with the retrieval uses a "safe" method.

with

- * At least one of the following is true:
 1. The HTTP request associated with the retrieval uses a "safe" method.
 2. The cookie's same-site-flag is "Default" and the amount of time elapsed since the cookie's creation-time is at most a duration of the user agent's choosing.

5.5. Storage Model

The user agent stores the following fields about each cookie: name, value, expiry-time, domain, path, creation-time, last-access-time, persistent-flag, host-only-flag, secure-only-flag, http-only-flag, and same-site-flag.

When the user agent "receives a cookie" from a request-uri with name cookie-name, value cookie-value, and attributes cookie-attribute-list, the user agent MUST process the cookie as follows:

1. A user agent MAY ignore a received cookie in its entirety. See Section 5.3.

2. If cookie-name is empty and cookie-value is empty, abort these steps and ignore the cookie entirely.
3. If the cookie-name or the cookie-value contains a %x00-08 / %x0A-1F / %x7F character (CTL characters excluding HTAB), abort these steps and ignore the cookie entirely.
4. If the sum of the lengths of cookie-name and cookie-value is more than 4096 octets, abort these steps and ignore the cookie entirely.
5. Create a new cookie with name cookie-name, value cookie-value. Set the creation-time and the last-access-time to the current date and time.
6. If the cookie-attribute-list contains an attribute with an attribute-name of "Max-Age":
 1. Set the cookie's persistent-flag to true.
 2. Set the cookie's expiry-time to attribute-value of the last attribute in the cookie-attribute-list with an attribute-name of "Max-Age".

Otherwise, if the cookie-attribute-list contains an attribute with an attribute-name of "Expires" (and does not contain an attribute with an attribute-name of "Max-Age"):

1. Set the cookie's persistent-flag to true.
2. Set the cookie's expiry-time to attribute-value of the last attribute in the cookie-attribute-list with an attribute-name of "Expires".

Otherwise:

1. Set the cookie's persistent-flag to false.
 2. Set the cookie's expiry-time to the latest representable date.
7. If the cookie-attribute-list contains an attribute with an attribute-name of "Domain":
 1. Let the domain-attribute be the attribute-value of the last attribute in the cookie-attribute-list with both an attribute-name of "Domain" and an attribute-value whose length is no more than 1024 octets. (Note that a leading

%x2E ("."), if present, is ignored even though that character is not permitted, but a trailing %x2E ("."), if present, will cause the user agent to ignore the attribute.)

Otherwise:

1. Let the domain-attribute be the empty string.
8. If the domain-attribute contains a character that is not in the range of [USASCII] characters, abort these steps and ignore the cookie entirely.
9. If the user agent is configured to reject "public suffixes" and the domain-attribute is a public suffix:

1. If the domain-attribute is identical to the canonicalized request-host:

1. Let the domain-attribute be the empty string.

Otherwise:

1. Abort these steps and ignore the cookie entirely.

NOTE: This step prevents attacker.example from disrupting the integrity of site.example by setting a cookie with a Domain attribute of "example".

10. If the domain-attribute is non-empty:
 1. If the canonicalized request-host does not domain-match the domain-attribute:
 1. Abort these steps and ignore the cookie entirely.
 - Otherwise:
 1. Set the cookie's host-only-flag to false.
 2. Set the cookie's domain to the domain-attribute.

Otherwise:

1. Set the cookie's host-only-flag to true.
2. Set the cookie's domain to the canonicalized request-host.

11. If the cookie-attribute-list contains an attribute with an attribute-name of "Path", set the cookie's path to attribute-value of the last attribute in the cookie-attribute-list with both an attribute-name of "Path" and an attribute-value whose length is no more than 1024 octets. Otherwise, set the cookie's path to the default-path of the request-uri.
12. If the cookie-attribute-list contains an attribute with an attribute-name of "Secure", set the cookie's secure-only-flag to true. Otherwise, set the cookie's secure-only-flag to false.
13. 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 cookie entirely.
14. If the cookie-attribute-list contains an attribute with an attribute-name of "HttpOnly", set the cookie's http-only-flag to true. Otherwise, set the cookie's http-only-flag to false.
15. If the cookie was received from a "non-HTTP" API and the cookie's http-only-flag is true, abort these steps and ignore the cookie entirely.
16. If the cookie's secure-only-flag is false, and the scheme component of request-uri does not denote a "secure" protocol, then abort these steps and ignore the 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 true.
 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'.

17. If the cookie-attribute-list contains an attribute with an attribute-name of "SameSite", and an attribute-value of "Strict", "Lax", or "None", set the cookie's same-site-flag to the attribute-value of the last attribute in the cookie-attribute-list with an attribute-name of "SameSite". Otherwise, set the cookie's same-site-flag to "Default".
18. If the cookie's same-site-flag is not "None":
 1. If the cookie was received from a "non-HTTP" API, and the API was called from a browsing context's active document whose "site for cookies" is not same-site with the top-level origin, then abort these steps and ignore the newly created cookie entirely.
 2. If the cookie was received from a "same-site" request (as defined in Section 5.2), skip the remaining substeps and continue processing the cookie.
 3. If the cookie was received from a request which is navigating a top-level browsing context [HTML] (e.g. if the request's "reserved client" is either null or an environment whose "target browsing context" is a top-level browsing context), skip the remaining substeps and continue processing the cookie.

Note: Top-level navigations can create a cookie with any SameSite value, even if the new cookie wouldn't have been sent along with the request had it already existed prior to the navigation.
 4. Abort these steps and ignore the newly created cookie entirely.
19. If the cookie's "same-site-flag" is "None", abort these steps and ignore the cookie entirely unless the cookie's secure-only-flag is true.
20. If the cookie-name begins with a case-sensitive match for the string "__Secure-", abort these steps and ignore the cookie entirely unless the cookie's secure-only-flag is true.
21. If the cookie-name begins with a case-sensitive match for the string "__Host-", abort these steps and ignore the cookie entirely unless the cookie meets all the following criteria:
 1. The cookie's secure-only-flag is true.

2. The cookie's host-only-flag is true.
 3. The cookie-attribute-list contains an attribute with an attribute-name of "Path", and the cookie's path is /.
22. If the cookie store contains a cookie with the same name, domain, host-only-flag, and path as the newly-created cookie:
1. Let old-cookie be the existing cookie with the same name, domain, host-only-flag, and path as the newly-created cookie. (Notice that this algorithm maintains the invariant that there is at most one such cookie.)
 2. If the newly-created cookie was received from a "non-HTTP" API and the old-cookie's http-only-flag is true, abort these steps and ignore the newly created cookie entirely.
 3. Update the creation-time of the newly-created cookie to match the creation-time of the old-cookie.
 4. Remove the old-cookie from the cookie store.
23. Insert the newly-created cookie into the cookie store.

A cookie is "expired" if the cookie has an expiry date in the past.

The user agent MUST evict all expired cookies from the cookie store if, at any time, an expired cookie exists in the cookie store.

At any time, the user agent MAY "remove excess cookies" from the cookie store if the number of cookies sharing a domain field exceeds some implementation-defined upper bound (such as 50 cookies).

At any time, the user agent MAY "remove excess cookies" from the cookie store if the cookie store exceeds some predetermined upper bound (such as 3000 cookies).

When the user agent removes excess cookies from the cookie store, the user agent MUST evict cookies in the following priority order:

1. Expired cookies.
2. Cookies whose secure-only-flag is false, 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.

If two cookies have the same removal priority, the user agent MUST evict the cookie with the earliest last-access-time first.

When "the current session is over" (as defined by the user agent), the user agent MUST remove from the cookie store all cookies with the persistent-flag set to false.

5.6. Retrieval Model

This section defines how cookies are retrieved from a cookie store in the form of a cookie-string. A "retrieval" is any event which requires generating a cookie-string. For example, a retrieval may occur in order to build a Cookie header field for an HTTP request, or may be required in order to return a cookie-string from a call to a "non-HTTP" API that provides access to cookies. A retrieval has an associated URI, same-site status, and type, which are defined below depending on the type of retrieval.

5.6.1. The Cookie Header Field

The user agent includes stored cookies in the Cookie HTTP request header field.

When the user agent generates an HTTP request, the user agent MUST NOT attach more than one Cookie header field.

A user agent MAY omit the Cookie header field in its entirety. For example, the user agent might wish to block sending cookies during "third-party" requests from setting cookies (see Section 7.1).

If the user agent does attach a Cookie header field to an HTTP request, the user agent MUST compute the cookie-string following the algorithm defined in Section 5.6.3, where the retrieval's URI is the request-uri, the retrieval's same-site status is computed for the HTTP request as defined in Section 5.2, and the retrieval's type is "HTTP".

5.6.2. Non-HTTP APIs

The user agent MAY implement "non-HTTP" APIs that can be used to access stored cookies.

A user agent MAY return an empty cookie-string in certain contexts, such as when a retrieval occurs within a third-party context (see Section 7.1).

If a user agent does return cookies for a given call to a "non-HTTP" API with an associated Document, then the user agent MUST compute the cookie-string following the algorithm defined in Section 5.6.3, where the retrieval's URI is defined by the caller (see [DOM-DOCUMENT-COOKIE]), the retrieval's same-site status is "same-site" if the Document's "site for cookies" is same-site with the top-level origin as defined in Section 5.2.1 (otherwise it is "cross-site"), and the retrieval's type is "non-HTTP".

5.6.3. Retrieval Algorithm

Given a cookie store and a retrieval, the following algorithm returns a cookie-string from a given cookie store.

1. Let cookie-list be the set of cookies from the cookie store that meets all of the following requirements:

- * Either:

- The cookie's host-only-flag is true and the canonicalized host of the retrieval's URI is identical to the cookie's domain.

Or:

- The cookie's host-only-flag is false and the canonicalized host of the retrieval's URI domain-matches the cookie's domain.

- * The retrieval's URI's path path-matches the cookie's path.

- * If the cookie's secure-only-flag is true, then the retrieval's URI's scheme must denote a "secure" protocol (as defined by the user agent).

NOTE: The notion of a "secure" protocol is not defined by this document. Typically, user agents consider a protocol secure if the protocol makes use of transport-layer security, such as SSL or TLS. For example, most user agents consider "https" to be a scheme that denotes a secure protocol.

- * If the cookie's http-only-flag is true, then exclude the cookie if the retrieval's type is "non-HTTP".
- * If the cookie's same-site-flag is not "None" and the retrieval's same-site status is "cross-site", then exclude the cookie unless all of the following conditions are met:

- The retrieval's type is "HTTP".
 - The same-site-flag is "Lax" or "Default".
 - The HTTP request associated with the retrieval uses a "safe" method.
 - The target browsing context of the HTTP request associated with the retrieval is a top-level browsing context.
2. The user agent SHOULD sort the cookie-list in the following order:
- * Cookies with longer paths are listed before cookies with shorter paths.
 - * Among cookies that have equal-length path fields, cookies with earlier creation-times are listed before cookies with later creation-times.
- NOTE: Not all user agents sort the cookie-list in this order, but this order reflects common practice when this document was written, and, historically, there have been servers that (erroneously) depended on this order.
3. Update the last-access-time of each cookie in the cookie-list to the current date and time.
4. Serialize the cookie-list into a cookie-string by processing each cookie in the cookie-list in order:
1. If the cookies' name is not empty, output the cookie's name followed by the %x3D ("=") character.
 2. If the cookies' value is not empty, output the cookie's value.
 3. If there is an unprocessed cookie in the cookie-list, output the characters %x3B and %x20 ("; ").

NOTE: Despite its name, the cookie-string is actually a sequence of octets, not a sequence of characters. To convert the cookie-string (or components thereof) into a sequence of characters (e.g., for presentation to the user), the user agent might wish to try using the UTF-8 character encoding [RFC3629] to decode the octet sequence. This decoding might fail, however, because not every sequence of octets is valid UTF-8.

6. Implementation Considerations

6.1. Limits

Practical user agent implementations have limits on the number and size of cookies that they can store. General-use user agents SHOULD provide each of the following minimum capabilities:

- * At least 50 cookies per domain.
- * At least 3000 cookies total.

User agents MAY limit the maximum number of cookies they store, and may evict any cookie at any time (whether at the request of the user or due to implementation limitations).

Note that a limit on the maximum number of cookies also limits the total size of the stored cookies, due to the length limits which MUST be enforced in Section 5.4.

Servers SHOULD use as few and as small cookies as possible to avoid reaching these implementation limits and to minimize network bandwidth due to the Cookie header field being included in every request.

Servers SHOULD gracefully degrade if the user agent fails to return one or more cookies in the Cookie header field because the user agent might evict any cookie at any time.

6.2. Application Programming Interfaces

One reason the Cookie and Set-Cookie header fields use such esoteric syntax is that many platforms (both in servers and user agents) provide a string-based application programming interface (API) to cookies, requiring application-layer programmers to generate and parse the syntax used by the Cookie and Set-Cookie header fields, which many programmers have done incorrectly, resulting in interoperability problems.

Instead of providing string-based APIs to cookies, platforms would be well-served by providing more semantic APIs. It is beyond the scope of this document to recommend specific API designs, but there are clear benefits to accepting an abstract "Date" object instead of a serialized date string.

6.3. IDNA Dependency and Migration

IDNA2008 [RFC5890] supersedes IDNA2003 [RFC3490]. However, there are differences between the two specifications, and thus there can be differences in processing (e.g., converting) domain name labels that have been registered under one from those registered under the other. There will be a transition period of some time during which IDNA2003-based domain name labels will exist in the wild. User agents SHOULD implement IDNA2008 [RFC5890] and MAY implement [UTS46] or [RFC5895] in order to facilitate their IDNA transition. If a user agent does not implement IDNA2008, the user agent MUST implement IDNA2003 [RFC3490].

7. Privacy Considerations

Cookies' primary privacy risk is their ability to correlate user activity. This can happen on a single site, but is most problematic when activity is tracked across different, seemingly unconnected Web sites to build a user profile.

Over time, this capability (warned against explicitly in [RFC2109] and all of its successors) has become widely used for varied reasons including:

- * authenticating users across sites,
- * assembling information on users,
- * protecting against fraud and other forms of undesirable traffic,
- * targeting advertisements at specific users or at users with specified attributes,
- * measuring how often ads are shown to users, and
- * recognizing when an ad resulted in a change in user behavior.

While not every use of cookies is necessarily problematic for privacy, their potential for abuse has become a widespread concern in the Internet community and broader society. In response to these concerns, user agents have actively constrained cookie functionality in various ways (as allowed and encouraged by previous specifications), while avoiding disruption to features they judge desirable for the health of the Web.

It is too early to declare consensus on which specific mechanism(s) should be used to mitigate cookies' privacy impact; user agents' ongoing changes to how they are handled are best characterised as experiments that can provide input into that eventual consensus.

Instead, this document describes limited, general mitigations against the privacy risks associated with cookies that enjoy wide deployment at the time of writing. It is expected that implementations will continue to experiment and impose stricter, more well-defined limitations on cookies over time. Future versions of this document might codify those mechanisms based upon deployment experience. If functions that currently rely on cookies can be supported by separate, targeted mechanisms, they might be documented in separate specifications and stricter limitations on cookies might become feasible.

Note that cookies are not the only mechanism that can be used to track users across sites, so while these mitigations are necessary to improve Web privacy, they are not sufficient on their own.

7.1. Third-Party Cookies

A "third-party" or cross-site cookie is one that is associated with embedded content (such as scripts, images, stylesheets, frames) that is obtained from a different server than the one that hosts the primary resource (usually, the Web page that the user is viewing). Third-party cookies are often used to correlate users' activity on different sites.

Because of their inherent privacy issues, most user agents now limit third-party cookies in a variety of ways. Some completely block third-party cookies by refusing to process third-party Set-Cookie header fields and refusing to send third-party Cookie header fields. Some partition cookies based upon the first-party context, so that different cookies are sent depending on the site being browsed. Some block cookies based upon user agent cookie policy and/or user controls.

While this document does not endorse or require a specific approach, it is RECOMMENDED that user agents adopt a policy for third-party cookies that is as restrictive as compatibility constraints permit. Consequently, resources cannot rely upon third-party cookies being treated consistently by user agents for the foreseeable future.

7.2. Cookie Policy

User agents MAY enforce a cookie policy consisting of restrictions on how cookies may be used or ignored (see Section 5.3).

A cookie policy may govern which domains or parties, as in first and third parties (see Section 7.1), for which the user agent will allow cookie access. The policy can also define limits on cookie size, cookie expiry (see Section 4.1.2.1 and Section 4.1.2.2), and the number of cookies per domain or in total.

The recommended cookie expiry upper limit is 400 days. User agents may set a lower limit to enforce shorter data retention timelines, or set the limit higher to support longer retention when appropriate (e.g., server-to-server communication over HTTPS).

The goal of a restrictive cookie policy is often to improve security or privacy. User agents often allow users to change the cookie policy (see Section 7.3).

7.3. User Controls

User agents SHOULD provide users with a mechanism for managing the cookies stored in the cookie store. For example, a user agent might let users delete all cookies received during a specified time period or all the cookies related to a particular domain. In addition, many user agents include a user interface element that lets users examine the cookies stored in their cookie store.

User agents SHOULD provide users with a mechanism for disabling cookies. When cookies are disabled, the user agent MUST NOT include a Cookie header field in outbound HTTP requests and the user agent MUST NOT process Set-Cookie header fields in inbound HTTP responses.

User agents MAY offer a way to change the cookie policy (see Section 7.2).

User agents MAY provide users the option of preventing persistent storage of cookies across sessions. When configured thusly, user agents MUST treat all received cookies as if the persistent-flag were set to false. Some popular user agents expose this functionality via "private browsing" mode [Aggarwal2010].

7.4. Expiration Dates

Although servers can set the expiration date for cookies to the distant future, most user agents do not actually retain cookies for multiple decades. Rather than choosing gratuitously long expiration periods, servers SHOULD promote user privacy by selecting reasonable cookie expiration periods based on the purpose of the cookie. For example, a typical session identifier might reasonably be set to expire in two weeks.

8. Security Considerations

8.1. Overview

Cookies have a number of security pitfalls. This section overviews a few of the more salient issues.

In particular, cookies encourage developers to rely on ambient authority for authentication, often becoming vulnerable to attacks such as cross-site request forgery [CSRF]. Also, when storing session identifiers in cookies, developers often create session fixation vulnerabilities.

Transport-layer encryption, such as that employed in HTTPS, is insufficient to prevent a network attacker from obtaining or altering a victim's cookies because the cookie protocol itself has various vulnerabilities (see "Weak Confidentiality" and "Weak Integrity", below). In addition, by default, cookies do not provide confidentiality or integrity from network attackers, even when used in conjunction with HTTPS.

8.2. Ambient Authority

A server that uses cookies to authenticate users can suffer security vulnerabilities because some user agents let remote parties issue HTTP requests from the user agent (e.g., via HTTP redirects or HTML forms). When issuing those requests, user agents attach cookies even if the remote party does not know the contents of the cookies, potentially letting the remote party exercise authority at an unwary server.

Although this security concern goes by a number of names (e.g., cross-site request forgery, confused deputy), the issue stems from cookies being a form of ambient authority. Cookies encourage server operators to separate designation (in the form of URLs) from authorization (in the form of cookies). Consequently, the user agent might supply the authorization for a resource designated by the attacker, possibly causing the server or its clients to undertake actions designated by the attacker as though they were authorized by the user.

Instead of using cookies for authorization, server operators might wish to consider entangling designation and authorization by treating URLs as capabilities. Instead of storing secrets in cookies, this approach stores secrets in URLs, requiring the remote entity to supply the secret itself. Although this approach is not a panacea, judicious application of these principles can lead to more robust security.

8.3. Clear Text

Unless sent over a secure channel (such as TLS), the information in the Cookie and Set-Cookie header fields is transmitted in the clear.

1. All sensitive information conveyed in these header fields is exposed to an eavesdropper.
2. A malicious intermediary could alter the header fields as they travel in either direction, with unpredictable results.
3. A malicious client could alter the Cookie header fields before transmission, with unpredictable results.

Servers SHOULD encrypt and sign the contents of cookies (using whatever format the server desires) when transmitting them to the user agent (even when sending the cookies over a secure channel). However, encrypting and signing cookie contents does not prevent an attacker from transplanting a cookie from one user agent to another or from replaying the cookie at a later time.

In addition to encrypting and signing the contents of every cookie, servers that require a higher level of security SHOULD use the Cookie and Set-Cookie header fields only over a secure channel. When using cookies over a secure channel, servers SHOULD set the Secure attribute (see Section 4.1.2.5) for every cookie. If a server does not set the Secure attribute, the protection provided by the secure channel will be largely moot.

For example, consider a webmail server that stores a session identifier in a cookie and is typically accessed over HTTPS. If the server does not set the Secure attribute on its cookies, an active network attacker can intercept any outbound HTTP request from the user agent and redirect that request to the webmail server over HTTP. Even if the webmail server is not listening for HTTP connections, the user agent will still include cookies in the request. The active network attacker can intercept these cookies, replay them against the server, and learn the contents of the user's email. If, instead, the server had set the Secure attribute on its cookies, the user agent would not have included the cookies in the clear-text request.

8.4. Session Identifiers

Instead of storing session information directly in a cookie (where it might be exposed to or replayed by an attacker), servers commonly store a nonce (or "session identifier") in a cookie. When the server receives an HTTP request with a nonce, the server can look up state information associated with the cookie using the nonce as a key.

Using session identifier cookies limits the damage an attacker can cause if the attacker learns the contents of a cookie because the nonce is useful only for interacting with the server (unlike non-nonce cookie content, which might itself be sensitive). Furthermore, using a single nonce prevents an attacker from "splicing" together cookie content from two interactions with the server, which could cause the server to behave unexpectedly.

Using session identifiers is not without risk. For example, the server SHOULD take care to avoid "session fixation" vulnerabilities. A session fixation attack proceeds in three steps. First, the attacker transplants a session identifier from his or her user agent to the victim's user agent. Second, the victim uses that session identifier to interact with the server, possibly imbuing the session identifier with the user's credentials or confidential information. Third, the attacker uses the session identifier to interact with server directly, possibly obtaining the user's authority or confidential information.

8.5. Weak Confidentiality

Cookies do not provide isolation by port. If a cookie is readable by a service running on one port, the cookie is also readable by a service running on another port of the same server. If a cookie is writable by a service on one port, the cookie is also writable by a service running on another port of the same server. For this reason, servers SHOULD NOT both run mutually distrusting services on different ports of the same host and use cookies to store security-sensitive information.

Cookies do not provide isolation by scheme. Although most commonly used with the http and https schemes, the cookies for a given host might also be available to other schemes, such as ftp and gopher. Although this lack of isolation by scheme is most apparent in non-HTTP APIs that permit access to cookies (e.g., HTML's document.cookie API), the lack of isolation by scheme is actually present in requirements for processing cookies themselves (e.g., consider retrieving a URI with the gopher scheme via HTTP).

Cookies do not always provide isolation by path. Although the network-level protocol does not send cookies stored for one path to another, some user agents expose cookies via non-HTTP APIs, such as HTML's document.cookie API. Because some of these user agents (e.g., web browsers) do not isolate resources received from different paths, a resource retrieved from one path might be able to access cookies stored for another path.

8.6. Weak Integrity

Cookies do not provide integrity guarantees for sibling domains (and their subdomains). For example, consider `foo.site.example` and `bar.site.example`. The `foo.site.example` server can set a cookie with a Domain attribute of `"site.example"` (possibly overwriting an existing `"site.example"` cookie set by `bar.site.example`), and the user agent will include that cookie in HTTP requests to `bar.site.example`. In the worst case, `bar.site.example` will be unable to distinguish this cookie from a cookie it set itself. The `foo.site.example` server might be able to leverage this ability to mount an attack against `bar.site.example`.

Even though the Set-Cookie header field supports the Path attribute, the Path attribute does not provide any integrity protection because the user agent will accept an arbitrary Path attribute in a Set-Cookie header field. For example, an HTTP response to a request for `http://site.example/foo/bar` can set a cookie with a Path attribute of `"/qux"`. Consequently, servers SHOULD NOT both run mutually distrusting services on different paths of the same host and use cookies to store security-sensitive information.

An active network attacker can also inject cookies into the Cookie header field sent to `https://site.example/` by impersonating a response from `http://site.example/` and injecting a Set-Cookie header field. The HTTPS server at `site.example` will be unable to distinguish these cookies from cookies that it set itself in an HTTPS response. An active network attacker might be able to leverage this ability to mount an attack against `site.example` even if `site.example` uses HTTPS exclusively.

Servers can partially mitigate these attacks by encrypting and signing the contents of their cookies, or by naming the cookie with the `__Secure-` prefix. However, using cryptography does not mitigate the issue completely because an attacker can replay a cookie he or she received from the authentic `site.example` server in the user's session, with unpredictable results.

Finally, an attacker might be able to force the user agent to delete cookies by storing a large number of cookies. Once the user agent reaches its storage limit, the user agent will be forced to evict some cookies. Servers SHOULD NOT rely upon user agents retaining cookies.

8.7. Reliance on DNS

Cookies rely upon the Domain Name System (DNS) for security. If the DNS is partially or fully compromised, the cookie protocol might fail to provide the security properties required by applications.

8.8. SameSite Cookies

8.8.1. 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.

8.8.2. 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 cookie-based 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://site.example/>. They might expect that clicking on an emailed link to <https://projects.example/secret/> project would show them the secret project that they're authorized to see, but if https://projects.example has marked their session cookies as SameSite=Strict, then this cross-site navigation won't send them along with the request. <https://projects.example> 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 conceptually granting "read" access, another granting "write" access. The latter could be marked as SameSite=Strict, and its absence would prompt a reauthentication step before executing any non-idempotent action.

The former could be marked as `SameSite=Lax`, in order to allow users access to data via top-level navigation, or `SameSite=None`, to permit access in all contexts (including cross-site embedded contexts).

8.8.3. Mashups and Widgets

The `Lax` and `Strict` values for the `SameSite` attribute are inappropriate for some important use-cases. In particular, note that content intended for embedding in cross-site contexts (social networking widgets or commenting services, for instance) will not have access to same-site cookies. Cookies which are required in these situations should be marked with `SameSite=None` to allow access in cross-site contexts.

Likewise, some forms of Single-Sign-On might require cookie-based authentication in a cross-site context; these mechanisms will not function as intended with same-site cookies and will also require `SameSite=None`.

8.8.4. Server-controlled

`SameSite` cookies in and of themselves don't do anything to address the general privacy concerns outlined in Section 7.1 of [RFC6265]. The "`SameSite`" 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 (for example, connection and/or socket pooling between same-site and cross-site requests).

8.8.5. Reload navigations

Requests issued for reloads triggered through user interface elements (such as a refresh button on a toolbar) are same-site only if the reloaded document was originally navigated to via a same-site request. This differs from the handling of other reload navigations, which are always same-site if top-level, since the source browsing context's active document is precisely the document being reloaded.

This special handling of reloads triggered through a user interface element avoids sending SameSite cookies on user-initiated reloads if they were withheld on the original navigation (i.e., if the initial navigation were cross-site). If the reload navigation were instead considered same-site, and sent all the initially withheld SameSite cookies, the security benefits of withholding the cookies in the first place would be nullified. This is especially important given that the absence of SameSite cookies withheld on a cross-site navigation request may lead to visible site breakage, prompting the user to trigger a reload.

For example, suppose the user clicks on a link from `https://attacker.example/` to `https://victim.example/`. This is a cross-site request, so `SameSite=Strict` cookies are withheld. Suppose this causes `https://victim.example/` to appear broken, because the site only displays its sensitive content if a particular SameSite cookie is present in the request. The user, frustrated by the unexpectedly broken site, presses refresh on their browser's toolbar. To now consider the reload request same-site and send the initially withheld SameSite cookie would defeat the purpose of withholding it in the first place, as the reload navigation triggered through the user interface may replay the original (potentially malicious) request. Thus, the reload request should be considered cross-site, like the request that initially navigated to the page.

8.8.6. Top-level requests with "unsafe" methods

The "Lax" enforcement mode described in Section 5.4.7.1 allows a cookie to be sent with a cross-site HTTP request if and only if it is a top-level navigation with a "safe" HTTP method. Implementation experience shows that this is difficult to apply as the default behavior, as some sites may rely on cookies not explicitly specifying a SameSite attribute being included on top-level cross-site requests with "unsafe" HTTP methods (as was the case prior to the introduction of the SameSite attribute).

For example, a login flow may involve a cross-site top-level POST request to an endpoint which expects a cookie with login information. For such a cookie, "Lax" enforcement is not appropriate, as it would cause the cookie to be excluded due to the unsafe HTTP request method. On the other hand, "None" enforcement would allow the cookie to be sent with all cross-site requests, which may not be desirable due to the cookie's sensitive contents.

The "Lax-allowing-unsafe" enforcement mode described in Section 5.4.7.2 retains some of the protections of "Lax" enforcement (as compared to "None") while still allowing cookies to be sent cross-site with unsafe top-level requests.

As a more permissive variant of "Lax" mode, "Lax-allowing-unsafe" mode necessarily provides fewer protections against CSRF. Ultimately, the provision of such an enforcement mode should be seen as a temporary, transitional measure to ease adoption of "Lax" enforcement by default.

9. IANA Considerations

9.1. Cookie

The permanent message header field registry (see [RFC3864]) needs to be updated with the following registration:

Header field name: Cookie

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document: this specification (Section 5.6.1)

9.2. Set-Cookie

The permanent message header field registry (see [RFC3864]) needs to be updated with the following registration:

Header field name: Set-Cookie

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document: this specification (Section 5.4)

9.3. Cookie Attribute Registry

IANA is requested to create the "Cookie Attribute Registry", defining the name space of attribute used to control cookies' behavior. The registry should be maintained at <https://www.iana.org/assignments/cookie-attribute-names> (<https://www.iana.org/assignments/cookie-attribute-names>).

9.3.1. Procedure

Each registered attribute name is associated with a description, and a reference detailing how the attribute is to be processed and stored.

New registrations happen on a "RFC Required" basis (see Section 4.7 of [RFC8126]). The attribute to be registered MUST match the extension-av syntax defined in Section 4.1.1. Note that attribute names are generally defined in CamelCase, but technically accepted case-insensitively.

9.3.2. Registration

The "Cookie Attribute Registry" should be created with the registrations below:

Name	Reference
Domain	Section 4.1.2.3 of this document
Expires	Section 4.1.2.1 of this document
HttpOnly	Section 4.1.2.6 of this document
Max-Age	Section 4.1.2.2 of this document
Path	Section 4.1.2.4 of this document
SameSite	Section 4.1.2.7 of this document
Secure	Section 4.1.2.5 of this document

Table 1

10. References

10.1. Normative References

- [DOM-DOCUMENT-COOKIE] WHATWG, "HTML - Living Standard", 18 May 2021, <<https://html.spec.whatwg.org/#dom-document-cookie>>.
- [FETCH] van Kesteren, A., "Fetch", n.d., <<https://fetch.spec.whatwg.org/>>.

- [HTML] Hickson, I., Pieters, S., van Kesteren, A., Jägenstedt, P., and D. Denicola, "HTML", n.d., <<https://html.spec.whatwg.org/>>.
- [HTTPSEM] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-19, 12 September 2021, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-semantics-19>>.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <<https://www.rfc-editor.org/rfc/rfc1034>>.
- [RFC1123] Braden, R., Ed., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, DOI 10.17487/RFC1123, October 1989, <<https://www.rfc-editor.org/rfc/rfc1123>>.
- [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/rfc/rfc2119>>.
- [RFC3490] Costello, A., "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003, <<https://www.rfc-editor.org/rfc/rfc3490>>. See Section 6.3 for an explanation why the normative reference to an obsoleted specification is needed.
- [RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, DOI 10.17487/RFC4790, March 2007, <<https://www.rfc-editor.org/rfc/rfc4790>>.
- [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/rfc/rfc5234>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<https://www.rfc-editor.org/rfc/rfc5890>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<https://www.rfc-editor.org/rfc/rfc6454>>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/rfc/rfc8126>>.

[SAMESITE] WHATWG, "HTML - Living Standard", 26 January 2021, <<https://html.spec.whatwg.org/#same-site>>.

[SERVICE-WORKERS] Russell, A., Song, J., and J. Archibald, "Service Workers", n.d., <<http://www.w3.org/TR/service-workers/>>.

[USASCII] American National Standards Institute, "Coded Character Set -- 7-bit American Standard Code for Information Interchange", ANSI X3.4, 1986.

10.2. Informative References

[Aggarwal2010] Aggarwal, G., Burzstein, E., Jackson, C., and D. Boneh, "An Analysis of Private Browsing Modes in Modern Browsers", 2010, <http://www.usenix.org/events/sec10/tech/full_papers/Aggarwal.pdf>.

[app-isolation] Chen, E., Bau, J., Reis, C., Barth, A., and C. Jackson, "App Isolation - Get the Security of Multiple Browsers with Just One", 2011, <<http://www.collinjackson.com/research/papers/appisolation.pdf>>.

[CSRF] Barth, A., Jackson, C., and J. Mitchell, "Robust Defenses for Cross-Site Request Forgery", DOI 10.1145/1455770.1455782, ISBN 978-1-59593-810-7, ACM CCS '08: Proceedings of the 15th ACM conference on Computer and communications security (pages 75-88), October 2008, <<http://portal.acm.org/citation.cfm?id=1455770.1455782>>.

[I-D.ietf-httpbis-cookie-alone] West, M., "Deprecate modification of 'secure' cookies from non-secure origins", Work in Progress, Internet-Draft, draft-ietf-httpbis-cookie-alone-01, 5 September 2016, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cookie-alone-01>>.

- [I-D.ietf-httpbis-cookie-prefixes]
West, M., "Cookie Prefixes", Work in Progress, Internet-Draft, draft-ietf-httpbis-cookie-prefixes-00, 23 February 2016, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cookie-prefixes-00>>.
- [I-D.ietf-httpbis-cookie-same-site]
West, M. and M. Goodwin, "Same-Site Cookies", Work in Progress, Internet-Draft, draft-ietf-httpbis-cookie-same-site-00, 20 June 2016, <<https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-cookie-same-site-00>>.
- [prerendering]
Bentzel, C., "Chrome Prerendering", n.d., <<https://www.chromium.org/developers/design-documents/prerender>>.
- [PSL] "Public Suffix List", n.d., <<https://publicsuffix.org/list/>>.
- [RFC2109] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2109, DOI 10.17487/RFC2109, February 1997, <<https://www.rfc-editor.org/rfc/rfc2109>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<https://www.rfc-editor.org/rfc/rfc2818>>.
- [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <<https://www.rfc-editor.org/rfc/rfc3629>>.
- [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/rfc/rfc3864>>.
- [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/rfc/rfc3986>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/rfc/rfc4648>>.

- [RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, DOI 10.17487/RFC5895, September 2010, <<https://www.rfc-editor.org/rfc/rfc5895>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<https://www.rfc-editor.org/rfc/rfc6265>>.
- [RFC7034] Ross, D. and T. Gondrom, "HTTP Header Field X-Frame-Options", RFC 7034, DOI 10.17487/RFC7034, October 2013, <<https://www.rfc-editor.org/rfc/rfc7034>>.
- [UTS46] Davis, M. and M. Suignard, "Unicode IDNA Compatibility Processing", UNICODE Unicode Technical Standards # 46, June 2016, <<http://unicode.org/reports/tr46/>>.

Appendix A. Changes

A.1. draft-ietf-httpbis-rfc6265bis-00

- * Port [RFC6265] to Markdown. No (intentional) normative changes.

A.2. draft-ietf-httpbis-rfc6265bis-01

- * Fixes to formatting caused by mistakes in the initial port to Markdown:
 - <https://github.com/httpwg/http-extensions/issues/243>
(<https://github.com/httpwg/http-extensions/issues/243>)
 - <https://github.com/httpwg/http-extensions/issues/246>
(<https://github.com/httpwg/http-extensions/issues/246>)
- * Addresses errata 3444 by updating the path-value and extension-av grammar, errata 4148 by updating the day-of-month, year, and time grammar, and errata 3663 by adding the requested note.
https://www.rfc-editor.org/errata_search.php?rfc=6265
(https://www.rfc-editor.org/errata_search.php?rfc=6265)
- * Dropped Cookie2 and Set-Cookie2 from the IANA Considerations section: <https://github.com/httpwg/http-extensions/issues/247>
(<https://github.com/httpwg/http-extensions/issues/247>)
- * Merged the recommendations from [I-D.ietf-httpbis-cookie-alone], removing the ability for a non-secure origin to set cookies with a 'secure' flag, and to overwrite cookies whose 'secure' flag is true.

- * Merged the recommendations from [I-D.ietf-httpbis-cookie-prefixes], adding `__Secure-` and `__Host-` cookie name prefix processing instructions.

A.3. draft-ietf-httpbis-rfc6265bis-02

- * Merged the recommendations from [I-D.ietf-httpbis-cookie-same-site], adding support for the `SameSite` attribute.
- * Closed a number of editorial bugs:
 - Clarified address bar behavior for `SameSite` cookies:
<https://github.com/httpwg/http-extensions/issues/201>
(<https://github.com/httpwg/http-extensions/issues/201>)
 - Added the word "Cookies" to the document's name:
<https://github.com/httpwg/http-extensions/issues/204>
(<https://github.com/httpwg/http-extensions/issues/204>)
 - Clarified that the `__Host-` prefix requires an explicit `Path` attribute: <https://github.com/httpwg/http-extensions/issues/222>
(<https://github.com/httpwg/http-extensions/issues/222>)
 - Expanded the options for dealing with third-party cookies to include a brief mention of partitioning based on first-party:
<https://github.com/httpwg/http-extensions/issues/248>
(<https://github.com/httpwg/http-extensions/issues/248>)
 - Noted that double-quotes in cookie values are part of the value, and are not stripped: <https://github.com/httpwg/http-extensions/issues/295> (<https://github.com/httpwg/http-extensions/issues/295>)
 - Fixed the "site for cookies" algorithm to return something that makes sense: <https://github.com/httpwg/http-extensions/issues/302> (<https://github.com/httpwg/http-extensions/issues/302>)

A.4. draft-ietf-httpbis-rfc6265bis-03

- * Clarified handling of invalid `SameSite` values:
<https://github.com/httpwg/http-extensions/issues/389>
(<https://github.com/httpwg/http-extensions/issues/389>)

- * Reflect widespread implementation practice of including a cookie's host-only-flag when calculating its uniqueness:
<https://github.com/httpwg/http-extensions/issues/199>
(<https://github.com/httpwg/http-extensions/issues/199>)
- * Introduced an explicit "None" value for the SameSite attribute:
<https://github.com/httpwg/http-extensions/issues/788>
(<https://github.com/httpwg/http-extensions/issues/788>)

A.5. draft-ietf-httpbis-rfc6265bis-04

- * Allow SameSite cookies to be set for all top-level navigations.
<https://github.com/httpwg/http-extensions/issues/594>
(<https://github.com/httpwg/http-extensions/issues/594>)
- * Treat Set-Cookie: token as creating the cookie ("", "token"):
<https://github.com/httpwg/http-extensions/issues/159>
(<https://github.com/httpwg/http-extensions/issues/159>)
- * Reject cookies with neither name nor value (e.g. Set-Cookie: = and Set-Cookie:: <https://github.com/httpwg/http-extensions/issues/159> (<https://github.com/httpwg/http-extensions/issues/159>))
- * Clarified behavior of multiple SameSite attributes in a cookie string: <https://github.com/httpwg/http-extensions/issues/901>
(<https://github.com/httpwg/http-extensions/issues/901>)

A.6. draft-ietf-httpbis-rfc6265bis-05

- * Typos and editorial fixes: <https://github.com/httpwg/http-extensions/pull/1035> (<https://github.com/httpwg/http-extensions/pull/1035>), <https://github.com/httpwg/http-extensions/pull/1038> (<https://github.com/httpwg/http-extensions/pull/1038>), <https://github.com/httpwg/http-extensions/pull/1040> (<https://github.com/httpwg/http-extensions/pull/1040>), <https://github.com/httpwg/http-extensions/pull/1047> (<https://github.com/httpwg/http-extensions/pull/1047>).

A.7. draft-ietf-httpbis-rfc6265bis-06

- * Editorial fixes: <https://github.com/httpwg/http-extensions/issues/1059> (<https://github.com/httpwg/http-extensions/issues/1059>), <https://github.com/httpwg/http-extensions/issues/1158> (<https://github.com/httpwg/http-extensions/issues/1158>).

- * Created a registry for cookie attribute names:
<https://github.com/httpwg/http-extensions/pull/1060>
(<https://github.com/httpwg/http-extensions/pull/1060>).
- * Tweaks to ABNF for cookie-pair and the Cookie header production:
<https://github.com/httpwg/http-extensions/issues/1074>
(<https://github.com/httpwg/http-extensions/issues/1074>),
<https://github.com/httpwg/http-extensions/issues/1119>
(<https://github.com/httpwg/http-extensions/issues/1119>).
- * Fixed serialization for nameless/valueless cookies:
<https://github.com/httpwg/http-extensions/pull/1143>
(<https://github.com/httpwg/http-extensions/pull/1143>).
- * Converted a normative reference to Mozilla's Public Suffix List [PSL] into an informative reference: <https://github.com/httpwg/http-extensions/issues/1159> (<https://github.com/httpwg/http-extensions/issues/1159>).

A.8. draft-ietf-httpbis-rfc6265bis-07

- * Moved instruction to ignore cookies with empty cookie-name and cookie-value from Section 5.4 to Section 5.5 to ensure that they apply to cookies created without parsing a cookie string:
<https://github.com/httpwg/http-extensions/issues/1234>
(<https://github.com/httpwg/http-extensions/issues/1234>).
- * Add a default enforcement value to the same-site-flag, equivalent to "SameSite=Lax": <https://github.com/httpwg/http-extensions/pull/1325> (<https://github.com/httpwg/http-extensions/pull/1325>).
- * Require a Secure attribute for "SameSite=None":
<https://github.com/httpwg/http-extensions/pull/1323>
(<https://github.com/httpwg/http-extensions/pull/1323>).
- * Consider scheme when running the same-site algorithm:
<https://github.com/httpwg/http-extensions/pull/1324>
(<https://github.com/httpwg/http-extensions/pull/1324>).

A.9. draft-ietf-httpbis-rfc6265bis-08

- * Define "same-site" for reload navigation requests, e.g. those triggered via user interface elements: <https://github.com/httpwg/http-extensions/pull/1384> (<https://github.com/httpwg/http-extensions/pull/1384>)

- * Consider redirects when defining same-site:
<https://github.com/httpwg/http-extensions/pull/1348>
(<https://github.com/httpwg/http-extensions/pull/1348>)
- * Align on using HTML terminology for origins:
<https://github.com/httpwg/http-extensions/pull/1416>
(<https://github.com/httpwg/http-extensions/pull/1416>)
- * Modify cookie parsing and creation algorithms in Section 5.4 and Section 5.5 to explicitly handle control characters:
<https://github.com/httpwg/http-extensions/pull/1420>
(<https://github.com/httpwg/http-extensions/pull/1420>)
- * Refactor cookie retrieval algorithm to support non-HTTP APIs:
<https://github.com/httpwg/http-extensions/pull/1428>
(<https://github.com/httpwg/http-extensions/pull/1428>)
- * Define "Lax-allowing-unsafe" SameSite enforcement mode:
<https://github.com/httpwg/http-extensions/pull/1435>
(<https://github.com/httpwg/http-extensions/pull/1435>)
- * Consistently use "header field" (vs 'header'):
<https://github.com/httpwg/http-extensions/pull/1527>
(<https://github.com/httpwg/http-extensions/pull/1527>)

A.10. draft-ietf-httpbis-rfc6265bis-09

- * Update cookie size requirements: <https://github.com/httpwg/http-extensions/pull/1563> (<https://github.com/httpwg/http-extensions/pull/1563>)
- * Reject cookies with control characters: <https://github.com/httpwg/http-extensions/pull/1576> (<https://github.com/httpwg/http-extensions/pull/1576>)
- * No longer treat horizontal tab as a control character:
<https://github.com/httpwg/http-extensions/pull/1589>
(<https://github.com/httpwg/http-extensions/pull/1589>)
- * Specify empty domain attribute handling:
<https://github.com/httpwg/http-extensions/pull/1709>
(<https://github.com/httpwg/http-extensions/pull/1709>)

A.11. draft-ietf-httpbis-rfc6265bis-10

- * Standardize Max-Age/Expires upper bound:
<https://github.com/httpwg/http-extensions/pull/1732>
(<https://github.com/httpwg/http-extensions/pull/1732>)

Acknowledgements

RFC 6265 was written by Adam Barth. This document is an update of RFC 6265, adding features and aligning the specification with the reality of today's deployments. Here, we're standing upon the shoulders of a giant since the majority of the text is still Adam's.

Authors' Addresses

Lily Chen (editor)
Google LLC
Email: chlily@google.com

Steven Englehardt (editor)
Mozilla
Email: senglehardt@mozilla.com

Mike West (editor)
Google LLC
Email: mkwst@google.com
URI: <https://mikewest.org/>

John Wilander (editor)
Apple, Inc
Email: wilander@apple.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 3, 2017

PH. Kamp
The Varnish Cache Project
October 30, 2016

HTTP header common structure
draft-kamp-httpbis-structure-01

Abstract

An abstract data model for HTTP headers, "Common Structure", and a HTTP/1 serialization of it, generalized from current HTTP headers.

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 3, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

The HTTP protocol does not impose any structure or datamodel on the information in HTTP headers, the HTTP/1 serialization is the datamodel: An ASCII string without control characters.

HTTP header definitions specify how the string must be formatted and while families of similar headers exist, it still requires an uncomfortable large number of bespoke parser and validation routines to process HTTP traffic correctly.

In order to improve performance HTTP/2 and HPACK uses naive text-compression, which incidentally decoupled the on-the-wire serialization from the data model.

During the development of HPACK it became evident that significantly bigger gains were available if semantic compression could be used, most notably with timestamps. However, the lack of a common data structure for HTTP headers would make semantic compression one long list of special cases.

Parallel to this, various proposals for how to fulfill data-transportation needs, and to a lesser degree to impose some kind of order on HTTP headers, at least going forward were floated.

All of these proposals, JSON, CBOR etc. run into the same basic problem: Their serialization is incompatible with [RFC7230]'s ABNF definition of 'field-value'.

For binary formats, such as CBOR, a wholesale base64/85 reserialization would be needed, with negative results for both debugability and bandwidth.

For textual formats, such as JSON, the format must first be neutered to not violate field-value's ABNF, and then workarounds added to reintroduce the features just lost, for instance UNICODE strings, and suddenly it is no longer JSON anymore.

This proposal starts from the other end, and builds and generalizes a data structure definition from existing HTTP headers, which means that HTTP/1 serialization and 'field-value' compatibility is built in.

If all future HTTP headers are defined to fit into this Common Structure we have at least halted the proliferation of bespoke parsers and started to pave the road for semantic compression serializations of HTTP traffic.

1.1. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

2. Definition of HTTP header Common Structure

The data model of Common Structure is an ordered sequence of named dictionaries. Please see Appendix A for how this model was derived.

The definition of the data model is on purpose abstract, uncoupled from any protocol serialization or programming environment representation, meant as the foundation on which all such manifestations of the model can be built.

Common Structure in ABNF:

```
import token from RFC7230
import DIGIT from RFC5234

common-structure = 1* ( identifier dictionary )

dictionary = * ( identifier value )

value = identifier /
        number /
        ascii_string /
        unicode_string /
        blob /
        timestamp /
        common-structure

identifier = token [ "/" token ]

number = ["-"] 1*15 DIGIT
        # XXX: Not sure how to do this in ABNF:
        # XXX: A single "." allowed between any two digits
        # The range is limited is to ensure it can be
        # correctly represented in IEEE754 64 bit
        # binary floating point format.

ascii_string = * %x20-7e
        # This is a "safe" string in the sense that it
        # contains no control characters or multi-byte
        # sequences. If that is not fancy enough, use
        # unicode_string.

unicode_string = * unicode_codepoint
        # XXX: Is there a place to import this from ?
        # Unrestricted unicode, because there is no sane
        # way to restrict or otherwise make unicode "safe".

blob = * %0x00-ff
        # Intended for cryptographic data and as a general
        # escape mechanism for unmet requirements.

timestamp = POSIX time_t with optional millisecond resolution
        # XXX: Is there a place to import this from ?
```

3. HTTP/1 serialization of HTTP header Common Structure

In ABNF:

```
import OWS from {{RFC7230}}
import HEXDIG, DQUOTE from {{RFC5234}}
```

```

import UTF8-2, UTF8-3, UTF8-4 from {{RFC3629}}

hl_common-structure-header =
  ( field-name ":" OWS ">" hl_common_structure "<" )
    # Self-identifying HTTP headers
  ( field-name ":" OWS hl_common_structure ) /
    # legacy HTTP headers on white-list, see {{iana}}

hl_common_structure = hl_element * ( "," hl_element )

hl_element = identifier * ( ";" identifier [ "=" hl_value ] )

hl_value = identifier /
  number /
  hl_ascii_string /
  hl_unicode_string /
  hl_blob /
  hl_timestamp /
  hl_common-structure

hl_ascii_string = DQUOTE *(
  ( "\" DQUOTE ) /
  ( "\" "\"" ) /
  0x20-21 /
  0x23-5B /
  0x5D-7E
  ) DQUOTE
  # This is a proper subset of hl_unicode_string
  # NB only allowed backslash escapes are \" and \\

hl_unicode_string = DQUOTE *(
  ( "\" DQUOTE )
  ( "\" "\"" ) /
  ( "\" "u" 4*HEXDIG ) /
  0x20-21 /
  0x23-5B /
  0x5D-7E /
  UTF8-2 /
  UTF8-3 /
  UTF8-4
  ) DQUOTE
  # This is UTF8 with HTTP1 unfriendly codepoints
  # (00-1f, 7f) neutered with \uXXXX escapes.

hl_blob = "'" base64 "'"
  # XXX: where to import base64 from ?

hl_timestamp = number

```

```
# UNIX/POSIX time_t semantics.  
# fractional seconds allowed.
```

```
hl_common_structure = ">" hl_common_structure "<"
```

XXX: Allow OWS in parsers, but not in generators ?

In programming environments which do not define a native representation or serialization of Common Structure, the HTTP/1 serialization should be used.

4. When to use Common Structure parser

All future standardized and all private HTTP headers using Common Structure should self identify as such. In the HTTP/1 serialization by making the first character ">" and the last "<". (These two characters are deliberately "the wrong way" to not clash with exsisting usages.)

Legacy HTTP headers which fit into Common Structure, are marked as such in the IANA Message Header Registry (see {iana}), and a snapshot of the registry can be used to trigger parsing according to Common Structure of these headers.

5. Desired normative effects

All new HTTP headers SHOULD use the Common Structure if at all possible.

6. Open/Outstanding issues to resolve

6.1. Single/multiple headers

Should we allow splitting common structure data over multiple headers ?

Pro:

Avoids size restrictions, easier on-the-fly editing

Contra:

Cannot act on any such header until all headers have been received.

We must define where headers can be split (between identifier and dictionary ?, in the middle of dictionaries ?)

Most on-the-fly editing is hackish at best.

7. Future work

7.1. Redefining existing headers for better performance

The HTTP/1 serializations self-identification mechanism makes it possible to extend the definition of existing Appendix C headers into Common Structure.

For instance one could imagine:

Date: >1475061449.201<

Which would be faster to parse and validate than the current definition of the Date header and more precise too.

Some kind of signal/negotiation mechanism would be required to make this work in practice.

7.2. Define a validation dictionary

A machine-readable specification of the legal contents of HTTP headers would go a long way to improve efficiency and security in HTTP implementations.

8. IANA considerations

The IANA Message Header Registry will be extended with an additional field named "Common Structure" which can have the values "True", "False" or "Unknown".

The RFC723x headers listed in Appendix B will get the value "True" in the new field.

The RFC723x headers listed in Appendix C will get the value "False" in the new field.

All other existing entries in the registry will be set to "Unknown" until and if the owner of the entry requests otherwise.

9. 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>>.

- [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>>.
- [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>>.

Appendix A. Does HTTP headers have any common structure ?

Several proposals have been floated in recent years to use some preexisting structured data serialization or other for HTTP headers, to impose some sanity.

None of these proposals have gained traction and no obvious candidate data serializations have been left unexamined.

This effort tries to tackle the question from the other side, by asking if there is a common structure in existing HTTP headers we can generalize for this purpose.

A.1. Survey of HTTP header structure

The RFC723x family of HTTP/1 standards control 49 entries in the IANA Message Header Registry, and they share two common motifs.

The majority of RFC723x HTTP headers are lists. A few of them are ordered, ('Content-Encoding'), some are unordered ('Connection') and some are ordered by 'q=%f' weight parameters ('Accept')

In most cases, the list elements are some kind of identifier, usually derived from ABNF 'token' as defined by [RFC7230].

A subgroup of headers, mostly related to MIME, uses what one could call a 'qualified token'::

```
qualified_token = token_or_asterix [ "/" token_or_asterix ]
```

The second motif is parameterized list elements. The best known is the "q=0.5" weight parameter, but other parameters exist as well.

Generalizing from these motifs, our candidate "Common Structure" data model becomes an ordered list of named dictionaries.

In pidgin ABNF, ignoring white-space for the sake of clarity, the HTTP/1.1 serialization of Common Structure is something like:

```
token_or_asterix = token from {{RFC7230}}, but also allowing "*"
qualified_token = token_or_asterix [ "/" token_or_asterix ]
field-name, see {{RFC7230}}

Common_Structure_Header = field-name ":" 1#named_dictionary
named_dictionary = qualified_token [ *(";" param) ]
param = token [ "=" value ]

value = we'll get back to this in a moment.
```

Nineteen out of the RFC723x's 48 headers, almost 40%, can already be parsed using this definition, and none the rest have requirements which could not be met by this data model. See Appendix B and Appendix C for the full survey details.

A.2. Survey of values in HTTP headers

Surveying the datatypes of HTTP headers, standardized as well as private, the following picture emerges:

A.2.1. Numbers

Integer and floating point are both used. Range and precision is mostly unspecified in controlling documents.

Scientific notation (9.192631770e9) does not seem to be used anywhere.

The ranges used seem to be minus several thousand to plus a couple of billions, the high end almost exclusively being POSIX time_t timestamps.

A.2.2. Timestamps

RFC723x text format, but POSIX time_t represented as integer or floating point is not uncommon. ISO8601 have also been spotted.

A.2.3. Strings

The vast majority are pure ASCII strings, with either no escapes, %xx URL-like escapes or C-style back-slash escapes, possibly with the addition of \uxxxx UNICODE escapes.

Where non-ASCII character sets are used, they are almost always implicit, rather than explicit. UTF8 and ISO-8859-1[5] seem to be most common.

A.2.4. Binary blobs

Often used for cryptographic data. Usually in base64 encoding, sometimes "-"-quoted more often not. base85 encoding is also seen, usually quoted.

A.2.5. Identifiers

Seems to almost always fit in the RFC723x 'token' definition.

A.3. Is this actually a useful thing to generalize ?

The number one wishlist item seems to be UNICODE strings, with a big side order of not having to write a new parser routine every time somebody comes up with a new header.

Having a common parser would indeed be a good thing, and having an underlying data model which makes it possible define a compressed serialization, rather than rely on serialization to text followed by text compression (ie: HPACK) seems like a good idea too.

However, when using a datamodel and a parser general enough to transport useful data, it will have to be followed by a validation step, which checks that the data also makes sense.

Today validation, such as it is, is often done by the bespoke parsers.

This then is probably where the next big potential for improvement lies:

Ideally a machine readable "data dictionary" which makes it possibly to copy that text out of RFCs, run it through a code generator which spits out validation code which operates on the output of the common parser.

But history has been particularly unkind to that idea.

Most attempts studied as part of this effort, have sunk under complexity caused by reaching for generality, but where scope has been wisely limited, it seems to be possible.

So file that idea under "future work".

Appendix B. RFC723x headers with "common structure"

Accept	[RFC7231, Section 5.3.2]
Accept-Charset	[RFC7231, Section 5.3.3]
Accept-Encoding	[RFC7231, Section 5.3.4][RFC7694, Section 3]
Accept-Language	[RFC7231, Section 5.3.5]
Age	[RFC7234, Section 5.1]
Allow	[RFC7231, Section 7.4.1]
Connection	[RFC7230, Section 6.1]
Content-Encoding	[RFC7231, Section 3.1.2.2]
Content-Language	[RFC7231, Section 3.1.3.2]
Content-Length	[RFC7230, Section 3.3.2]
Content-Type	[RFC7231, Section 3.1.1.5]
Expect	[RFC7231, Section 5.1.1]
Max-Forwards	[RFC7231, Section 5.1.2]
MIME-Version	[RFC7231, Appendix A.1]
TE	[RFC7230, Section 4.3]
Trailer	[RFC7230, Section 4.4]
Transfer-Encoding	[RFC7230, Section 3.3.1]
Upgrade	[RFC7230, Section 6.7]
Vary	[RFC7231, Section 7.1.4]

Appendix C. RFC723x headers with "uncommon structure"

1 of the RFC723x headers is only reserved, and therefore have no structure at all:

Close	[RFC7230, Section 8.1]
-------	------------------------

5 of the RFC723x headers are HTTP dates:

Date	[RFC7231, Section 7.1.1.2]
Expires	[RFC7234, Section 5.3]
If-Modified-Since	[RFC7232, Section 3.3]
If-Unmodified-Since	[RFC7232, Section 3.4]
Last-Modified	[RFC7232, Section 2.2]

24 of the RFC723x headers use bespoke formats which only a single or in rare cases two headers share:

Accept-Ranges	[RFC7233, Section 2.3]
bytes-unit / other-range-unit	
Authorization	[RFC7235, Section 4.2]
Proxy-Authorization	[RFC7235, Section 4.4]
credentials	
Cache-Control	[RFC7234, Section 5.2]

1#cache-directive

Content-Location [RFC7231, Section 3.1.4.2]
absolute-URI / partial-URI

Content-Range [RFC7233, Section 4.2]
byte-content-range / other-content-range

ETag [RFC7232, Section 2.3]
entity-tag

Forwarded [RFC7239]
1#forwarded-element

From [RFC7231, Section 5.5.1]
mailbox

If-Match [RFC7232, Section 3.1]
If-None-Match [RFC7232, Section 3.2]
"*" / 1#entity-tag

If-Range [RFC7233, Section 3.2]
entity-tag / HTTP-date

Host [RFC7230, Section 5.4]
uri-host [":" port]

Location [RFC7231, Section 7.1.2]
URI-reference

Pragma [RFC7234, Section 5.4]
1#pragma-directive

Range [RFC7233, Section 3.1]
byte-ranges-specifier / other-ranges-specifier

Referer [RFC7231, Section 5.5.2]
absolute-URI / partial-URI

Retry-After [RFC7231, Section 7.1.3]
HTTP-date / delay-seconds

Server [RFC7231, Section 7.4.2]
User-Agent [RFC7231, Section 5.5.3]
product *(RWS (product / comment))

Via [RFC7230, Section 5.7.1]
1#(received-protocol RWS received-by [RWS comment])

Warning [RFC7234, Section 5.5]
1#warning-value

Proxy-Authenticate [RFC7235, Section 4.3]
WWW-Authenticate [RFC7235, Section 4.1]
1#challenge

Author's Address

Poul-Henning Kamp
The Varnish Cache Project

Email: phk@varnish-cache.org

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

K. Oku
DeNA Co., Ltd.
October 31, 2016

An HTTP Status Code for Indicating Hints
draft-kazuho-early-hints-status-code-00

Abstract

This memo introduces an informational status code for HTTP that can be used for indicating hints to help a client start making preparations for processing the final response.

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.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Notational Conventions	3
2. 103 Early Hints	3
3. Interoperability Issues	3
4. Security Considerations	3
5. IANA Considerations	3
6. References	4
6.1. Normative References	4
6.2. Informative References	4
Author's Address	4

1. Introduction

Most if not all of the web pages processed by a web browser contain links to external resources that need to be fetched prior to rendering the documents. Therefore, it is beneficial to send such links as early as possible in order to minimize the time spent until the browser becomes possible to render the document. Link header of type "preload" ([Preload]) can be used to indicate such links within the response headers of an HTTP response.

However, it is not always possible for an origin server to send a response immediately after receiving a request. In fact, it is often the contrary. There are many deployments in which an origin server needs to query a database before generating a response. It is also not unusual for an origin server to delegate a request to an upstream HTTP server running at a distant location.

The dilemma here is that even though it is preferable for an origin server to send some headers as soon as it receives a request, it cannot do so until the status code and the headers of the final HTTP response is determined.

HTTP/2 ([RFC7540]) push can be used as a solution to the issue, but has its own limitations. The resources that can be pushed using HTTP/2 are limited to those belonging to the same origin. Also, it is impossible to send only the links of the resources using HTTP/2 push. Sending HTTP responses for every resource is an inefficient way of using bandwidth, especially when a caching server exists as an intermediary.

This memo defines a status code for sending an informational response ([RFC7231], section 6.2) that contains headers that are likely to be included in the final response. A server can send the informational response containing some of the headers to help the client start making preparations for processing the final response, and then run

time-consuming operations to generate the final response. The informational response can also be used by an origin server to trigger HTTP/2 push at an caching intermediary.

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. 103 Early Hints

This informational status code indicates the client that the server is likely to send a final request with the headers included in the informational response.

A server MUST NOT include Content-Length, Transfer-Encoding, or any hop-by-hop headers ([RFC7230], section 6.1) in the informational response using the status code.

A client MAY speculatively evaluate the headers included in the informational response while waiting for the final response. For example, a client may recognize the link header of type preload and start fetching the resource. However, the evaluation MUST NOT affect how the final response is processed; the client must behave as if it had not seen the informational response.

An intermediary MAY drop the informational response. It MAY send HTTP/2 ([RFC7540]) push responses using the information found in the informational response.

3. Interoperatibility Issues

Clients may have issues handling Early Hints, since informational response is rarely used for requests not including an Expect header ([RFC7231], section 5.1.1). Therefore, it is desirable to negotiate the capability to use the status code.

4. Security Considerations

TBD

5. IANA Considerations

If Early Hints is standardized, the HTTP Status Codes Registry should be updated with the following entries:

- o Code: 103

- o Description: Early Hints
- o Specification: this document

6. References

6.1. 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>>.
- [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>>.
- [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>>.

6.2. Informative References

- [Preload] Grigorik, I., "Preload", September 2016, <<https://w3c.github.io/preload/>>.

Author's Address

Kazuho Oku
DeNA Co., Ltd.

Email: kazuhooku@gmail.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 1, 2017

J. Butler

W. Lee

B. McQuade

K. Mixter
October 28, 2016

A Proposal for Shared Dictionary Compression over HTTP
draft-lee-sdch-spec-00

Abstract

This paper proposes an HTTP/1.1-compatible extension that supports inter-response data compression by means of a reference dictionary shared between user agent and server.

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 1, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

In order to reduce payload size, HTTP/1.1 supports response compression via the Accept-Encoding and Content-Encoding headers. The most commonly used HTTP response compression encoding is gzip, which compresses data that is repeated within a given response. However, HTTP/1.1 does not provide a mechanism for compressing data that is repeated between responses. A different class of encoding technique, known as delta encoding, has proven effective at compressing inter-response data.

Previous efforts to extend HTTP/1.1 to support delta compression have focused on encoding an HTTP response as a delta of a previous version of that response. One such approach is discussed in RFC 3229 "Delta encoding in HTTP" [RFC3229]. While RFC 3229 is effective at reducing payload size for many types of resources, it may not be suitable for certain classes of responses.

Specifically, under RFC 3229, deltas can only be applied to responses originating from the same URL, and the means of identifying the instance to delta "from" is by a Last-Modified timestamp or entity-tag. This makes RFC 3229 unsuitable for compressing dynamically generated responses to a given URL with varying query parameters (e.g. a search results page), since these types of responses are difficult to identify uniquely using entity tags or last modified timestamps. Content hashes can be used, but false positives are possible. Also, storing all previous responses on the server may not be practical.

2. Proposal: Shared Dictionary Compression over HTTP

Existing techniques compress each response in isolation, and so cannot take advantage of cross-payload redundancy. For example, retrieving a set of HTML pages with the same header, footer, inlined JavaScript and CSS requires the retransmission of the same data multiple times. This paper proposes a compression technique that leverages this cross-payload redundancy.

In this proposal, a dictionary is a file downloaded by the user agent from the server that contains strings which are likely to appear in subsequent HTTP responses. In the case described above, if the header, footer, JavaScript and CSS are stored in a dictionary possessed by both user agent and server, the server can substitute these elements with references to the dictionary, and the user agent can reconstruct the original page from these references. By

substituting dictionary references for repeated elements in HTTP responses, the payload size can be reduced.

If either the user agent or the server does not support the extension, then ordinary HTTP responses are served.

If both the user agent and the server support the extension but the user agent does not have an applicable dictionary (as described in detail below), the server responds with an ordinary HTTP response that includes a header advertising the location of a relevant dictionary. This dictionary can be retrieved out-of-band by the user agent.

If both the user agent and the server support the extension and the user agent has an applicable dictionary, then each HTTP response includes references to strings in the dictionary, rather than repeating those strings in the response. The references require fewer bytes to encode than the strings themselves, reducing the payload size.

The HTTP header-based protocol for negotiating the presence of dictionaries on user agent and server is referred to in this proposal as the SDCH protocol. The compression scheme based on a particular dictionary shared between user agent and server is referred to as the SDCH encoding, and is built upon the VCDIFF compression data format [RFC3284].

3. Syntax

The grammar descriptions in the sections that follow depend on the following syntax: DIGIT (decimal digit), BASE64URLDIGIT (alphanumeric digit or "-" or "_"), PAYLOADBYTE (a byte), token (informally, a sequence of non-special, non-white space characters), rest-of-line (informally, a sequence of characters not including carriage return or line-feed). In the grammar below, HTTP_url, abs_path, and query are defined in RFC 7230 [RFC7230].

```
header = attr ":" value "\n"
attr = token
value = rest-of-line
dictionary-client-id = 1*BASE64URLDIGIT
dictionary-server-id = 1*BASE64URLDIGIT
payload = 1*PAYLOADBYTE
vcdiff-payload = 1*PAYLOADBYTE
partial-url = HTTP_url | abs_path [ "?" query ]
```

The attribute names (attr) are case-insensitive. White space is permitted between tokens.

4. Dictionary Description

4.1. General

In the proposed protocol, a dictionary can only be used with a limited set of URLs and for a limited duration of time, referred to as its scope and lifetime, respectively. A dictionary is composed of the data used by the compression algorithm, known as the payload, as well as metadata describing its scope and lifetime. The scope is specified by a domain attribute and path attribute that are patterned after the same named attributes from the HTTP State Management Specification [RFC2965].

4.2. Syntax of Dictionary Metadata

The syntax of dictionary metadata is as follows:

```
dictionary-metadata = 1*dictionary-header "\n"
dictionary-header = "domain" ":" value "\n"
                  | "path" ":" value "\n"
                  | "path-equals" ":" value "\n"
                  | "format-version" ":" value "\n"
                  | "max-age" ":" value "\n"
                  | "port" ":" <"> portlist <"> "\n"
portlist = 1#portnum
portnum = 1*DIGIT
```

A complete dictionary definition then has this format: n dictionary-definition = dictionary-metadata payload

Informally, the metadata for a dictionary is a series of headers, similar in form to HTTP headers, terminated by an empty line. The dictionary payload begins immediately after this blank line.

The valid dictionary header identifiers are described below:

- o Domain: domain.

Required. Indicates the domain to which the dictionary applies. The domain specification must explicitly start with a dot. For example, a dictionary with the domain specification ".google.com" may be used to compress a response served from the host name www.google.com, but not used to compress a response served from the host name www.gmail.com. Only printable ASCII characters are permitted in the domain value. International Domain Names must be specified using IDNA.

- o Path: path.

Optional. Indicates the set of URL paths for which this dictionary is valid. If unspecified, the dictionary applies to all paths within the given domain.

- o Path-equals: path.

Optional. Indicates the exact URL path for which this dictionary is valid. If both "path" and "path-equals" are specified, the dictionary applies only to those URLs which satisfy both criteria.

- o Format-version: version.

Optional. Indicates the version of the dictionary payload. If unspecified, the format version defaults to "1.0". Currently, the only acceptable value is "1.0".

- o Max-age: delta-seconds.

Optional. Indicates the amount of time that a dictionary can be advertised to the server by the user agent, relative to the time it was downloaded. If unspecified, the default is 30 days from the time the dictionary was downloaded by the user agent. * Port: port list. Optional. Indicates the comma-separated list of ports to which this dictionary applies. If unspecified, the dictionary applies to all ports.

Like HTTP headers, dictionary header identifiers are case-insensitive. Unknown headers will be ignored by the user agent, allowing other headers to be added in the future.

4.3. Dictionary Scope

The specific rules of when a dictionary can be applied to a URL, i.e. that define its scope, are modeled after the rules for cookie scoping. The term "domain-match" is defined in RFC 2965. We define path-matching as follows For two strings that represent paths, P1 and P2, P1 path-matches P2 if either:

1. P2 is equal to P1
2. P2 is a prefix of P1 and either the final character in P2 is "/" or the character following P2 in P1 is "/".

For example, "/tec/waldo" path-matches "/tec", "/tec/", and "/tec/waldo", but does not path-match "/tec/wal".

Given these definitions of domain-match and path-match, a request URL falls within a dictionary's scope exactly when all of the following are true:

1. The request URL's host name domain-matches the Domain attribute of the dictionary.
2. If the dictionary has a Port attribute, the request port is one of the ports listed in the Port attribute.
3. The request URL path-matches the path attribute of the dictionary.
4. The request URL's scheme matches the scheme of the dictionary.

If a URL falls within a dictionary's scope, the dictionary is said to "apply" to the URL.

4.4. Dictionary Identifier

In communications between user agent and server, a dictionary is identified by the first 96 bits of the SHA-256 digest [RFC6234] of a dictionary's metadata and payload (see dictionary-definition above) exactly as it is received by the user agent from the server. Both user agent and server compute this identifier independently, based on the metadata and the payload of the dictionary. This digest should be unique within a dictionary's scope (domain and path) in order to prevent dictionary identifier collisions.

The digest serves not only as an identifier but also as a safeguard against attempts to maliciously intercept or otherwise modify dictionary contents, since a compromised dictionary will hash to a different identifier and the server will not recognize it. The user agent identifier for a dictionary is defined as the URL-safe base64 encoding (as described in RFC 3548, section 4 [RFC3548] of the first 48 bits (bits 0..47) of the dictionary's SHA-256 digest. The server identifier for a dictionary is the URL-safe base64 encoding of the second 48 bits (bits 48..95). When identifying a dictionary to the server, the user agent uses the user agent identifier, and similarly, when identifying a dictionary to the user agent, the server uses the server identifier. Note that both user agent and server have the entire dictionary and can thus compute both identifiers for the dictionary.

As a consequence of this scheme, dictionaries do not need to be explicitly named by site maintainers, as the protocol avoids identifying them in any way other than the above digest-generated identifiers.

4.5. Differences between Dictionaries and Cookies

Dictionaries are similar to cookies in that they allow sharing of state over HTTP. Thus, we have modeled dictionaries after cookies, as described in RFC 2965. However, because dictionaries are typically larger than cookies, embedding a dictionary in the response would increase latency of the response. Thus a dictionary is always sent as a separate HTTP response (unlike a cookie which is included in a Set-Cookie header of any HTTP response). The Get-Dictionary HTTP response header is used to tell the user agent that it should fetch a dictionary separately for use in future requests.

Likewise, rather than including the dictionary contents in the HTTP request headers (like a cookie in the Cookie header), dictionary identifiers (described above) are used to advertise available dictionaries in HTTP requests from the user agent to the server.

5. User Agent / Server Interaction Description

5.1. User Agent Role in HTTP Request Generation

The user agent:

1. Advertises support for the proposed protocol by adding the "sdch" token to the Accept-Encoding header of HTTP requests.
2. Advertises any dictionaries it possesses that apply to the URL being requested (per the scoping rules above) in the Avail-Dictionary request header.

The Avail-Dictionary header syntax is as follows: avail-dictionary-header = "Avail-Dictionary" ":" 1#dictionary-client where dictionary-client-id is the user agent identifier part for the dictionary based on the SHA-256 digest as described above. The value of this header is informally a comma separated list of user agent dictionary identifiers.

The user agent must advertise every dictionary it has cached that applies to the requested URL. It is only the presence of the dictionary identifier in this header that indicates to the server that the user agent possesses and therefore does not need to download the dictionary. Since the user agent must advertise every dictionary it has, it is the site maintainer's responsibility to avoid making too many dictionaries available at a given time. Advertising many dictionaries in this header can counteract the benefits of compression.

Note that for each individual request the user agent has discretion over whether or not to add "sdch" Accept-Encoding token and the Avail-Dictionary header. Since some responses, such as image data, are unlikely to benefit from dictionary compression, the user agent can reduce the size of its requests by not sending this token and header. The user agent may decide whether or not to add these headers based on file extensions in URLs or the context of the request. For instance, the user agent may choose to not advertise SDCH for URLs referenced in IMG elements.

5.2. Server Role in HTTP Response Generation

When a server that supports the extension receives a request that indicates that the user agent supports the protocol (e.g. the "sdch" token is present in the Accept-Encoding request header), two independent decisions must be made. The server must decide: 1. if it wants to send an encoded response. 2. if it wants to inform the user agent about additional dictionaries it can download and use in the future.

The server may return an encoded response only if all of the following are true: 1. The Accept-Encoding request header contains the "sdch" token. 2. The server can send a response compressed with a dictionary whose dictionary-client-id is in the Avail-Dictionary request header.

A server may return a response that is not encoded even if it recognizes a dictionary advertised by the user agent. If the server decides to not use SDCH encoding when a Avail-Dictionary header is present, it must include a specific HTTP header X-SDCH-Encoding with value "0" in the response. The syntax of the X-SDCH-Encoding header is:

```
sdch-not-used-header = "X-SDCH-Encoding" ":" "0"
```

The server indicates that an HTTP response is encoded by inserting the token "sdch" into the Content-Encoding header of the HTTP response.

A compatible server may instruct a compatible user agent to download one or more new dictionaries by including the Get-Dictionary header in the HTTP response. The server may advertise a Get-Dictionary header even if the response is not encoded. The syntax of the Get-Dictionary header is: `get-dictionary-header = "Get-Dictionary" ":" "1#partial-url` where `partial-url` is either a complete URL, or just the absolute URL path (in which case the scheme, host, and port of the originating server would be used when requesting the dictionary). If a complete URL is provided, it must have the same scheme, host, and

port as the originating server. The Content-Type header of dictionary responses must be application/x-sdch-dictionary. The value in the get dictionary header is a comma-separated list of partial-url elements.

The server must not advertise a dictionary with a dictionary-client-id that the user agent has listed in the Avail-Dictionary header.

The server may use SDCH compression with a dictionary that the user agent has advertised and also include a Get-Dictionary header for a different dictionary that the user agent has not advertised.

The server must prevent SDCH-encoded responses from being cached by intermediate proxies. See the section below on proxy caching for additional details.

The server should limit the number of active dictionaries at any one time, by using well-scoped dictionaries. A server that has many active dictionaries with overlapping scope will cause user agents to generate a very long Avail-Dictionary header, the overhead of which can counteract the benefits of SDCH compression.

The server may decide to precompute and cache SDCH-encoded responses if a given SDCH-encoded response will be served multiple times (e.g. for static content).

The server may apply multiple Content-Encodings to the response, (e.g. sdch and gzip) in which case subsequent encoding tokens are appended to the Content-Encoding header, per the HTTP/1.1 RFC section 14.11.

5.3. User Agent Role in HTTP Response Handling

An SDCH-compatible user agent must inspect the Content-Encoding HTTP response header to determine if the response is SDCH-encoded. If the Content-Encoding includes the "sdch" token, the user agent must perform SDCH decompression on the response.

If the HTTP response includes a Get-Dictionary header, the user agent must verify that the partial-url specified refers to the same server that generated the response. If so, the user agent may download the dictionary at the given URL.

There are two different URLs to consider when downloading and storing a dictionary. The referer URL is the URL of the request that resulted in the server responding with a Get-Dictionary header.

The dictionary URL is defined as follows:

1. If the partial-url is a complete URL, the dictionary URL is the partial-url.
2. If the partial-url is just a path URL, the dictionary URL is generated from the scheme and host name of the referrer URL and the path in the partial-url.

The user agent may retrieve a dictionary if the origin of the dictionary matches the origin of the referrer. HTTP redirects may only be followed if the origin matches as well.

Upon retrieving the dictionary, the user agent must validate the dictionary. Here again, the validation rules are modeled after the rules for when a user agent can accept an HTTP cookie. A dictionary is invalid and must not be stored if any of the following are true:

1. The dictionary has no Domain attribute.
2. The effective host name that derives from the referrer URL host name does not domain-match the Domain attribute.
3. The Domain attribute is a top level domain.
4. The referrer URL host is a host domain name (not IP address) and has the form HD, where D is the value of the Domain attribute, and H is a string that contains one or more dots.
5. If the dictionary has a Port attribute and the referrer URL's port was not in the list.

If the dictionary is valid and user agent decides to store the dictionary, the scheme of the dictionary URL should also be stored along with dictionary.

5.4. SDCH-Encoded Response Body

An SDCH-encoded response starts with the dictionary-server-id used to compress the response. The syntax of the SDCH-encoded response is: dictionary-compression-response = dictionary-server-id "\0" vcdiff-payload

6. Examples

For the purpose of these examples, assume the following dictionaries exist on the server and can be downloaded from the following URLs:

"Search results" dictionary

- o domain: .google.com
 - o path: /search
 - o user agent ID: TWFuIGlz
 - o server ID: JOWk0d2N
 - o download location: /dictionaries/search_dict
- "Help pages" dictionary
- o domain: .google.com
 - o path: /
 - o user agent ID: GVhc3V48
 - o server ID: 09d2_m3-
 - o download location: /dictionaries/help_dict

Note that the dictionary identifier consists of two parts: user agent ID and the server ID. Most of the detail of the request and response headers has been omitted.

6.1. Example 1: Initial Interaction, User Agent has No Dictionaries

1. user agent's request

```
GET /search?q=sprouts HTTP/1.1
Host: www.google.com
Accept-Encoding: sdch, gzip
```

1. server's response

```
HTTP/1.1 200 OK
Content-type: text/html
Content-Encoding: gzip
Get-Dictionary: /dictionaries/search_dict, /dictionaries/help_dict
Cache-Control: private
```

Note that the response returned by the server does NOT use SDCH encoding, since the user agent does not have a dictionary. The server simply provides the locations of the dictionaries for future use. The user agent may choose to retrieve one or both dictionaries separately.

6.2. Example 2: User Agent Requests the Dictionary

1. user agent's request

```
GET /dictionaries/search_dict HTTP/1.1
Host: www.google.com
Accept-Encoding: sdch, gzip
```

1. server's response

```
HTTP/1.1 200 OK
Content-type: application/x-sdch-dictionary
Content-Encoding: gzip
```

```
Domain: .google.com
Path: /search
Format-version: 1.0
```

...dictionary contents...

Upon receiving this response, the user agent computes the digest of the dictionary and determines the user agent ID is TWFuIGlz and the server ID is JOWk0d2N.

6.3. Example 3: User Requests Page AND User Agent Has Already Downloaded

the Dictionary

1. user agent's request

```
GET /search&q=brussel+sprouts HTTP/1.1
Host: www.google.com
Accept-Encoding: sdch, gzip
Avail-Dictionary: TWFuIGlz
```

1. server's response

```
HTTP/1.1 200 OK
Content-type: text/html
Content-Encoding: sdch, gzip
Get-Dictionary: /dictionaries/help_dict
Cache-Control: private
```

JOWk0d2N<NUL>...VCDIFFed response...
(note that the response shown to the left the result of gzip decompression)

The server has properly identified the dictionary using its server ID and the user agent can confirm that the second 48 bits of the SHA-256 digest of the dictionary match its computation. It can then decompress the VCDIFF response using this dictionary. Even though the "search results" dictionary was used to decompress the response, the server has chosen to indicate another dictionary could be requested by the user agent from http://www.google.com/dictionaries/help_dict. This dictionary must be different than the "search results" dictionary as the server must never request the user agent download a dictionary it knows the user agent already has. Let's assume the user agent decides to download this dictionary.

6.4. Example 4: User Requests with Multiple Dictionaries

1. user agent's request

```
GET /search&q=brussels HTTP/1.1
Host: www.google.com
Accept-Encoding: sdch, gzip
Avail-Dictionary: GVhc3V48,TWFuIGlz
```

1. server's response

```
HTTP/1.1 200 OK
Content-type: text/html
Content-Encoding: sdch, gzip
Cache-Control: private
```

JOWk0d2N<NUL>...VCDIFFed response... (note that the response shown to the left the result of gzip decompression)

The user agent advertises that it has already downloaded two dictionaries that apply. The server may compress the response with either dictionary. As the server has no other dictionaries that apply to the request, it does not advertise any dictionaries in its response.

7. Implementation Considerations

7.1. Implementation Limits

There are practical limitations to the number and size of the dictionaries a user agent can store. It is suggested that general use, non-mobile user agents should have the following minimum capabilities:

- o At least 300 dictionaries stored

- o At least 100KB of payload per dictionary
- o At least 10MB of total dictionary contents
- o At least 20 dictionaries stored per domain

7.2. Dictionary Downloading

The user agent always has the choice of whether or not to download a dictionary. It is recommended that the user agent be implemented with sufficient state to avoid downloading too many dictionaries from the same server. A malfunctioning server may also request the user agent continually download the same dictionary. One simple method to avoid both of these possibilities is for the user agent to rate-limit downloading dictionaries from the same domain.

When the user agent receives a response with a Get-Dictionary header with dictionary download URLs that it may fetch, it should perform the dictionary downloads in the background. This is possible as the dictionary to be downloaded is guaranteed to not be needed to decompress the response with the Get-Dictionary header. The user agent should be careful to abort background dictionary downloads that do not complete in a reasonable amount of time.

7.3. Data Integrity

If the dictionaries are tied to individual users or specific user actions, HTTP may leak this information to passive attacker by allowing the Get-Dictionary info to be seen. When using HTTPS, the same risk is prevented in the design document since Get-Dictionary URLs are required to be same-origin as the response.

However, Downloading dictionaries over HTTPS or advertising dictionaries over HTTPS might introduce new security risks.

TODO: add some examples. For example, SDCH-over-HTTPS subject to compression oracle attacks similar to CRIME/BREACH with the difference that the compression context is not supplied by the attacker. If an attacker had the contents of a dictionary, there is a theoretical possibility where a server sends a static response XOR'ed with user-provided data. The Attacker can provide data which reduced the size of the response when XOR'ed with the static response, the attacker may then be able to determine the contents of the static response.

The protocol needs to ensure that the content as decompressed by the user agent with a given dictionary is identical to the server's

originally intended content. The three areas that can cause a data integrity problem are discussed below.

7.3.1. Data tampered by Proxy

We have found incorrectly implemented proxies which tamper with an SDCH response and make the response unable to be decompressed to the server's originally intended content. The tampering may not be detected in the SDCH encoding itself if the proxy makes SDCH content look like non-SDCH content, for instance, by stripping the 'sdch' token from the content-encoding header of the response or by adding additional encodings (like gzip) on top of the SDCH and gzipped response without making the Content-Encoding header match. In order to detect when this occurs, the HTTP header X-SDCH-Encoding must be added to the response by the server to inform the client that the response was originally not SDCH encoded by the server. Should the user agent advertise SDCH capability in the request but receive a non-SDCH encoded response without the X-SDCH-Encoding header, it suggests that the response was tampered by a proxy. The user agent may then take action to avoid using SDCH in the future.

7.3.2. Dictionary mismatch

When a dictionary information is exchanged between user agent and server, it is necessary to ensure that the dictionary identifiers are completely unambiguous, or the decompressed result may differ from the original content. To address this issue, SDCH uses the first 96 bits of the SHA-256 digest of a dictionary's metadata and payload to create the dictionary identifiers used by the user agent and server to avoid ambiguity. (Please refer to the section "Dictionaries description" above for details.)

7.3.3. Data corruption / malicious attacks

While this issue is not specific to SDCH, it can be exacerbated due to the nature of the stateful compression. For example, if the dictionary is corrupted or maliciously modified in a persistent on-disk cache, all subsequent responses decoded by using this dictionary will be corrupt. For this reason, the user agent and server should revalidate the dictionaries' integrity when they are loaded from non-volatile storage.

Other issues like data corruption during transmission in the encoded payload could have much bigger adverse effect than that in the plain text. TCP provides a checksum, but it cannot detect some errors like swapped bytes. To address this issue, SDCH includes an Adler32 checksum [RFC1950] in the encoded data shards. (Please refer to appendix "VCDIFF Encoding Format and SDCH" for details.)

8. Response Caching

8.1. User Agent Cache

The user agent should honor HTTP caching directives (Cache-Control, Expires,...) for caching responses, whether or not the responses are SDCH-encoded. When caching the SDCH-encoded responses, the SDCH-encoded responses should be decoded before being written to the cache. If this is not possible, the user agent may cache SDCH-encoded responses, unless the HTTP response headers indicate that the response is not cacheable. In this case, an SDCH-encoded cache entry should be invalidated when (1) the dictionary used to encode that response is deleted from the dictionary store, (2) the SDCH decompression user agent is uninstalled (if it is implemented as a browser add-on), or (3) the SDCH capable user agent is disabled.

Intermediate Caches

The server should use HTTP cache headers that prevent non-SDCH-aware intermediate cache servers from storing the encoded contents. The cache directive "Cache-Control: private" can be used for this purpose.

If the compressed response can be cached by proxy caches, the server must include the HTTP header "Vary: Accept-Encoding, Avail-Dictionary" to alert proxies about sending the cached content only to the user agents who can decode it. Note that some proxies may not respect the Vary header, in which case non-SDCH-capable user agents would end up downloading SDCH-encoded responses. Thus, we recommend that SDCH-encoded responses not be cacheable by intermediate proxies unless there is a very compelling reason. Further, "Vary: Accept-Encoding, Avail-Dictionary" will not match requests unless these headers match exactly.

A proxy cache may provide one of three levels of support for caching SDCH-encoded objects.

1. No support - Never cache any response if the header Vary is present.
2. Basic support - The proxy cache only serves cached SDCH-encoded content if all cache serving conditions are satisfied and the values of the HTTP headers specified in the Vary header of the cached content exactly match the corresponding headers in the HTTP request.
3. Full support - The proxy should understand the SDCH protocol, should know what dictionary is used to encode/decode the

response, and should be able to download advertised dictionaries. The cache needs to have both SDCH user agent and server logic in it. The server should store the SDCH decoded responses in its cache.

Dictionary Caching User Agent Cache

As dictionary payloads may be large compared to the size of individual HTTP responses, in order to maximize latency improvements and minimize the bandwidth overhead of downloading dictionaries, it is recommended that the user agent persistently store dictionaries in a dictionary cache (e.g. on disk). It is suggested that the user agent implement a maximum limit on number of dictionaries stored per domain in order to avoid allowing one domain to force dictionaries for other domains out of the user agent's dictionary cache. To implement a fixed maximum size cache it is recommended that the cache manager first evict the dictionaries that were least recently used for decoding.

Ideally dictionaries will be stored in the same cache as HTTP responses and may be inspected and cleared by the user using existing user interfaces. However, new support may be created to fulfill the need for the user agent to be able to quickly determine which dictionaries should be advertised for a given request.

The user agent should be careful to validate that a dictionary matches its original identifier before being used for decompression to prevent malicious attacks on the dictionary cache. The user agent may implicitly handle this by always recomputing the hash before advertising the dictionary. However, to improve efficiency, the user agent may cache the original digest of the dictionary, advertise the dictionary with that digest, and then only for the dictionary selected by the server to encode the response, verify that the cached dictionary digest still matches the digest computed from the cached dictionary.

The user agent must not evict dictionaries from its dictionary store that have been advertised in the Avail-Dictionary header of a HTTP request for which a response has not yet been returned.

If a user agent downloads a dictionary which has the same identifier as another previously downloaded dictionary which are applicable to the same hosts, the user agent must be careful to either ignore the new dictionary or evict the old dictionary. If the two dictionaries with the same identifier have exactly the same contents the choice is not important, however this indicates a server error as a server must never instruct the user agent to download a dictionary that was advertised by the user agent. The user agent may want to avoid

downloading dictionaries from this server in the future as they may not be new and downloading unnecessary dictionaries can increase latency.

Intermediate Caches

The dictionary should be treated as a regular HTTP response by intermediate proxies. Thus, the normal HTTP caching consideration for intermediate proxies should apply to the dictionary as well.

9. Future Directions

=====

As currently proposed, SDCH is not applicable to another case where differential compression would be beneficial: large files that change infrequently and in small ways, such as JavaScript and CSS files referenced by other HTML documents.

TODO: Re-evaluate dictionary scoping rules, current approach that patterned after the same named attributes from the HTTP State Management Specification [RFC2965] may not be the best choice.

10. Current Status and Updates

For current information about the status of this proposal:
<https://groups.google.com/group/SDCH>

11. IANA Considerations

This document makes no requests of IANA.

12. Security Considerations

Some security considerations are discussed in the data integrity section above, but the author anticipates further work to describe these.

13. Acknowledgements

The authors would like to acknowledge the support of Google, Inc. for the development of this work. Technical editor: Harriett Hardman. Feedback and comments: Greg Badros, Chandra Chereddi, Darren Fisher, Ted Hardie, Ashu Jain, Ian Hickson, Othman Laraki, Jim Roskind, Ryan Sleevi, Lincoln Smith, Randy Smith, and Linus Upson.

14. Appendix: VCDIFF Encoding Format and SDCH

Although the SDCH protocol is proposed so that it could be adapted for use with any differential-encoding format, it currently uses the VCDIFF encoding format. This format was chosen because its definition is publicly available as the RFC 3284 draft standard. The VCDIFF format is independent of the method used for finding the longest possible matches between the dictionary (source) data and the payload (target) data.

An encoder and decoder for the VCDIFF format, intended for use with SDCH, has been released as open-source under the Apache license. This package is called "open-vcdiff". It uses the Bentley/McIlroy technique for finding matches between the dictionary and target data. It conforms to the VCDIFF draft standard, with the following exceptions:

Interleaved format

The VCDIFF draft standard format divides each encoded delta window into three sections (data, instructions, and addresses), with the aim of improving compressibility of the encoded file using a secondary compressor such as gzip. The drawback to this approach is that none of the target data can be reconstructed unless the entire delta window is available. The delta window is received in packets over the network and it is desirable to be able to process its contents as they arrive. In order to facilitate decoding a stream of packets from the network, we have modified the VCDIFF format so that it interleaves the data, instructions, and addresses instead of placing them in three separate sections. Each instruction is followed by its size and then by an address or literal data.

Adler32 checksum

The format can be modified to include an Adler32 checksum [RFC1950] of the target window data. If the checksum format is used, then bit 2 (0x04, defined as VCD_CHECKSUM) of the Win_Indicator byte will be set, and the checksum will appear just after the "Length of addresses for COPYs" field and before the "Data section for ADDs and RUNs" section in the encoding.

Version header byte (Header4)

If either of the two enhancements described above is used, then the resulting format will not conform to the VCDIFF draft standard as described in RFC 3284. In order to indicate this deviation from the standard, the fourth byte in the encoding (Header4, reserved for the VCDIFF version code) will be set to 0x53 (a capital "S" character in

ASCII.) If neither enhancement is used, the fourth byte may be 0x00 (a null character), the default value described in the standard.

VCD_TARGET flag and target COPY instructions not allowed for SDCH

The SDCH protocol is intended to produce a delta between static dictionary data and target data. Secondary compression with gzip will be used to eliminate redundancy within the target data. For this reason, when using VCDIFF for SDCH, the Win_Indicator flag should always include the VCD_SOURCE flag, never the VCD_TARGET flag. COPY instructions should only reference addresses within the source data, never within the previously decoded target.

The Xdelta package (<http://xdelta.org>) produces a format based on VCDIFF, though not 100% compatible with the RFC draft standard. That package has been released under the GNU General Public License.

15. References

15.1. 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>>.
- [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>>.

15.2. Informative References

- [RFC3284] Korn, D., MacDonald, J., Mogul, J., and K. Vo, "The VCDIFF Generic Differencing and Compression Data Format", RFC 3284, DOI 10.17487/RFC3284, June 2002, <<http://www.rfc-editor.org/info/rfc3284>>.
- [RFC3229] Mogul, J., Krishnamurthy, B., Douglass, F., Feldmann, A., Goland, Y., van Hoff, A., and D. Hellerstein, "Delta encoding in HTTP", RFC 3229, DOI 10.17487/RFC3229, January 2002, <<http://www.rfc-editor.org/info/rfc3229>>.
- [RFC3929] Hardie, T., "Alternative Decision Making Processes for Consensus-Blocked Decisions in the IETF", RFC 3929, DOI 10.17487/RFC3929, October 2004, <<http://www.rfc-editor.org/info/rfc3929>>.

- [RFC3548] Josefsson, S., Ed., "The Base16, Base32, and Base64 Data Encodings", RFC 3548, DOI 10.17487/RFC3548, July 2003, <<http://www.rfc-editor.org/info/rfc3548>>.
- [RFC2965] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2965, DOI 10.17487/RFC2965, October 2000, <<http://www.rfc-editor.org/info/rfc2965>>.
- [RFC1950] Deutsch, P. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, DOI 10.17487/RFC1950, May 1996, <<http://www.rfc-editor.org/info/rfc1950>>.
- [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, <<http://www.rfc-editor.org/info/rfc6234>>.

Authors' Addresses

Jon Butler

Email: jkbutler@google.com

Wei-Hsin Lee

Email: weihsinl@google.com

Bryan McQuade

Email: mcquade@google.com

Kenneth Mixter

Email: kmixter@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: June 22, 2017

P. McManus
Mozilla
December 19, 2016

HTTP Immutable Responses
draft-mcmanus-immutable-01

Abstract

The immutable HTTP response Cache-Control extension allows servers to identify resources that will not be updated during their freshness lifetime. This assures that a client never needs to revalidate a cached fresh resource to be certain it has not been modified.

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 June 22, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

1. Introduction

The HTTP freshness lifetime [RFC7234] caching attribute specifies that a client may safely reuse a response to satisfy future requests over a specific period of time. It does not specify that the resource will be not be modified during that period.

For instance, a front page newspaper photo with a freshness lifetime of one hour would mean that no user should see a photo more than one hour old. However, the photo could be updated at any time resulting in different users seeing different photos depending on the contents of their caches for up to one hour. This is compliant with the caching mechanism defined in [RFC7234].

Users that need to confirm there have been no updates to their current cached resources typically invoke the reload (or refresh) mechanism in the user agent. This in turn generates a conditional request [RFC7232] and either a new representation or, if unmodified, a 304 response [RFC7231] is returned. A user agent that manages HTML and its dependent sub-resources may issue hundreds of conditional requests to refresh all portions of a common HTML page [REQPERPAGE].

Through the use of the versioned URL design pattern some content providers never create more than one variant of a sub-resource. When these resources need an update they are simply published under a new URL, typically embedding a variant identifier in the path, and references to the sub-resource are updated with the new path information.

For example, <https://www.example.com/101016/main.css> might be updated and republished as <https://www.example.com/102026/main.css> and the html that references it is changed at the same time. This design pattern allows a very large freshness lifetime to be applied to the sub-resource without guessing when it will be updated in the future.

Unfortunately, the user-agent is not aware of the versioned URL design pattern. User driven refresh events still translate into wasted conditional requests for each sub-resource as each will return 304 responses.

The immutable HTTP response Cache-Control extension allows servers to identify resources that will not be updated during their freshness lifetime. This effectively instructs the client that any conditional request for a previously served variant of that resource may be safely skipped without worrying that it has been updated.

2. The immutable Cache-Control extension

When present in an HTTP response, the immutable Cache-Control extension indicates that the origin server **MUST NOT** update the representation of that resource during the freshness lifetime of the response.

The immutable extension only applies during the freshness lifetime of the response. Stale responses **SHOULD** be revalidated as they normally would be in the absence of immutable.

The immutable extension takes no arguments and if any arguments are present they have no meaning. Multiple instances of the immutable extension are equivalent to one instance. The presence of an immutable Cache-Control extension in a request has no effect.

2.1. About Intermediaries

An immutable response has the same semantic meaning for proxy clients as it does for User-Agent based clients and they therefore **MAY** also presume a conditional revalidation for a response marked immutable would return 304. A proxy client who uses immutable to anticipate a 304 response may choose whether to reply with a 304 or 200 to its requesting client.

2.2. Example

Cache-Control: max-age=31536000, immutable

3. Security Considerations

The immutable mechanism acts as form of soft pinning and, as with all pinning mechanisms, creates a vector for amplification of cache corruption incidents. These incidents include cache poisoning attacks. Three mechanisms are suggested for mitigation of this risk:

- o Clients should ignore immutable for resources that are not part of an authenticated context such as HTTPS. Authenticated resources are less vulnerable to cache poisoning.
- o User-Agents often provide two different refresh mechanisms: reload and some form of force-reload. The latter is used to rectify interrupted loads and other corruption. These reloads, typically indicated through no-cache request attributes, should ignore immutable as well.

- o Clients should ignore immutable for resources that do not provide a strong indication that the stored response size is the correct response size such as responses delimited by connection close.

4. IANA Considerations

[RFC7234] sections 7.1 and 7.1.2 require registration of the immutable extension in the "Hypertext Transfer Protocol (HTTP) Cache Directive Registry" with IETF Review.

- o Cache-Directive: immutable
- o Pointer to specification text: [this document]

5. Acknowledgments

Thank you to Ben Maurer for partnership in developing and testing this idea. Thank you to Amos Jeffries for help with proxy interactions.

6. References

6.1. Normative References

- [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>>.
- [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>>.

6.2. Informative References

- [REQPERPAGE] "HTTP Archive", n.d., <<http://httparchive.org/interesting.php#reqTotal>>.

Author's Address

Patrick McManus
Mozilla

Email: pmcmanus@mozilla.com

Network Working Group
Internet-Draft
Intended status: Experimental
Expires: June 4, 2017

E. Stark
Google
December 01, 2016

Expect-CT Extension for HTTP
draft-stark-expect-ct-01

Abstract

This document defines a new HTTP header, named Expect-CT, that allows web host operators to instruct user agents to expect valid Signed Certificate Timestamps (SCTs) to be served on connections to these hosts. When configured in enforcement mode, user agents (UAs) will remember that hosts expect SCTs and will refuse connections that do not conform to the UA's Certificate Transparency policy. When configured in report-only mode, UAs will report the lack of valid SCTs to a URI configured by the host, but will allow the connection. By turning on Expect-CT, web host operators can discover misconfigurations in their Certificate Transparency deployments and ensure that misissued certificates accepted by UAs are discoverable in Certificate Transparency logs.

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 June 4, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Requirements Language	3
1.2. Terminology	3
2. Server and Client Behavior	4
2.1. Response Header Field Syntax	4
2.1.1. The report-uri Directive	5
2.1.2. The enforce Directive	6
2.1.3. The max-age Directive	6
2.2. Server Processing Model	7
2.2.1. HTTP-over-Secure-Transport Request Type	7
2.2.2. HTTP Request Type	7
2.3. User Agent Processing Model	7
2.3.1. Expect-CT Header Field Processing	7
2.3.2. Noting an Expect-CT Host - Storage Model	8
2.3.3. HTTP-Equiv <meta> Element Attribute	9
2.4. Noting Expect-CT	9
2.5. Evaluating Expect-CT Connections for CT Compliance	10
3. Reporting Expect-CT Failure	11
3.1. Generating a violation report	11
3.2. Sending a violation report	13
4. Security Considerations	13
4.1. Maximum max-age	13
5. Privacy Considerations	13
6. IANA Considerations	13
7. Usability Considerations	13
8. References	13
8.1. Normative References	13
8.2. URIs	15
Author's Address	15

1. Introduction

This document defines a new HTTP header that enables UAs to identify web hosts that expect the presence of Signed Certificate Timestamps (SCTs) [RFC6962] in future Transport Layer Security (TLS) [RFC5246] connections.

Web hosts that serve the Expect-CT HTTP header are noted by the UA as Known Expect-CT Hosts. The UA evaluates each connection to a Known Expect-CT Host for compliance with the UA's Certificate Transparency (CT) Policy. If the connection violates the CT Policy, the UA sends a report to a URI configured by the Expect-CT Host and/or fails the connection, depending on the configuration that the Expect-CT Host has chosen.

If misconfigured, Expect-CT can cause unwanted connection failures (for example, if a host deploys Expect-CT but then switches to a legitimate certificate that is not logged in Certificate Transparency logs, or if a web host operator believes their certificate to conform to all UAs' CT policies but is mistaken). Web host operators are advised to deploy Expect-CT with caution, by using the reporting feature and gradually increasing the interval where the UA remembers the host as a Known Expect-CT Host. These precautions can help web host operators gain confidence that their Expect-CT deployment is not causing unwanted connection failures.

Expect-CT is a trust-on-first-use (TOFU) mechanism. The first time a UA connects to a host, it lacks the information necessary to require SCTs for the connection. Thus, the UA will not be able to detect and thwart an attack on the UA's first connection to the host. Still, Expect-CT provides value by 1) allowing UAs to detect the use of unlogged certificates after the initial communication, and 2) allowing web hosts to be confident that UAs are only trusting publicly-auditable certificates.

1.1. Requirements Language

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 RFC 2119 [RFC2119].

1.2. Terminology

Terminology is defined in this section.

Certificate Transparency Policy is a policy defined by the UA concerning the number, sources, and delivery mechanisms of Signed Certificate Timestamps that are served on TLS connections. The policy defines the properties of a connection that must be met in order for the UA to consider it CT-qualified.

Certificate Transparency Qualified describes a TLS connection for which the UA has determined that a sufficient quantity and quality of Signed Certificate Timestamps have been provided.

CT-qualified See Certificate Transparency Qualified.

CT Policy See Certificate Transparency Policy.

Expect-CT Host See HTTP Expect-CT Host.

HTTP Expect-CT is the overall name for the combined UA- and server-side security policy defined by this specification.

HTTP Expect-CT Host is a conformant host implementing the HTTP server aspects of HTTP Expect-CT. This means that an Expect-CT Host returns the "Expect-CT" HTTP response header field in its HTTP response messages sent over secure transport.

Known Expect-CT Host is an Expect-CT Host that the UA has noted as such. See Section 2.4 for particulars.

UA is an acronym for "user agent". For the purposes of this specification, a UA is an HTTP client application typically actively manipulated by a user [RFC2616].

Unknown Expect-CT Host is an Expect-CT Host that the UA has not noted.

2. Server and Client Behavior

2.1. Response Header Field Syntax

The "Expect-CT" header field is a new response header defined in this specification. It is used by a server to indicate that UAs should evaluate connections to the host emitting the header for CT compliance (Section 2.5).

Figure 1 describes the syntax (Augmented Backus-Naur Form) of the header field, using the grammar defined in RFC 5234 [RFC5234] and the rules defined in Section 3.2 of RFC 7230 [RFC7230].

```
Expect-CT-Directives = directive *( OWS ";" OWS directive )
directive             = directive-name [ "=" directive-value ]
directive-name        = token
directive-value        = token / quoted-string
```

Figure 1: Syntax of the Expect-CT header field

Optional white space ("OWS") is used as defined in Section 3.2.3 of RFC 7230 [RFC7230]. "token" and "quoted-string" are used as defined in Section 3.2.6 of RFC 7230 [RFC7230].

The directives defined in this specification are described below. The overall requirements for directives are:

1. The order of appearance of directives is not significant.
2. A given directive MUST NOT appear more than once in a given header field. Directives are either optional or required, as stipulated in their definitions.
3. Directive names are case insensitive.
4. UAs MUST ignore any header fields containing directives, or other header field value data, that do not conform to the syntax defined in this specification. In particular, UAs must not attempt to fix malformed header fields.
5. If a header field contains any directive(s) the UA does not recognize, the UA MUST ignore those directives.
6. If the Expect-CT header field otherwise satisfies the above requirements (1 through 5), the UA MUST process the directives it recognizes.

2.1.1.1. The report-uri Directive

The OPTIONAL "report-uri" directive indicates the URI to which the UA SHOULD report Expect-CT failures (Section 2.5). The UA POSTs the reports to the given URI as described in Section 3.

The "report-uri" directive is REQUIRED to have a directive value, for which the syntax is defined in Figure 2.

report-uri-value = absolute-URI

Figure 2: Syntax of the report-uri directive value

"absolute-URI" is defined in Section 4.3 of RFC 3986 [RFC3986].

Hosts may set "report-uri"s that use HTTP or HTTPS. If the scheme in the "report-uri" is one that uses TLS (e.g., HTTPS), UAs MUST check Expect-CT compliance when the host in the "report-uri" is a Known Expect-CT Host; similarly, UAs MUST apply HSTS if the host in the "report-uri" is a Known HSTS Host.

Note that the report-uri need not necessarily be in the same Internet domain or web origin as the host being reported about.

UAs SHOULD make their best effort to report Expect-CT failures to the "report-uri", but they may fail to report in exceptional conditions. For example, if connecting the "report-uri" itself incurs an Expect-CT failure or other certificate validation failure, the UA MUST cancel the connection. Similarly, if Expect-CT Host A sets a "report-uri" referring to Expect-CT Host B, and if B sets a "report-uri" referring to A, and if both hosts fail to comply to the UA's CT Policy, the UA SHOULD detect and break the loop by failing to send reports to and about those hosts.

UAs SHOULD limit the rate at which they send reports. For example, it is unnecessary to send the same report to the same "report-uri" more than once.

2.1.1.2. The enforce Directive

The OPTIONAL "enforce" directive is a valueless directive that, if present (i.e., it is "asserted"), signals to the UA that compliance to the CT Policy should be enforced (rather than report-only) and that the UA should refuse future connections that violate its CT Policy. When both the "enforce" directive and "report-uri" directive (as defined in Figure 2) are present, the configuration is referred to as an "enforce-and-report" configuration, signalling to the UA both that compliance to the CT Policy should be enforced and that violations should be reported.

2.1.1.3. The max-age Directive

The "max-age" directive specifies the number of seconds after the reception of the Expect-CT header field during which the UA SHOULD regard the host from whom the message was received as a Known Expect-CT Host.

The "max-age" directive is REQUIRED to be present within an "Expect-CT" header field. The "max-age" directive is REQUIRED to have a directive value, for which the syntax (after quoted-string unescaping, if necessary) is defined in Figure 3.

```
max-age-value = delta-seconds
delta-seconds = 1*DIGIT
```

Figure 3: Syntax of the max-age directive value

"delta-seconds" is used as defined in Section 1.2.1 of RFC 7234 [RFC7234].

2.2. Server Processing Model

This section describes the processing model that Expect-CT Hosts implement. The model has 2 parts: (1) the processing rules for HTTP request messages received over a secure transport (e.g., authenticated, non-anonymous TLS); and (2) the processing rules for HTTP request messages received over non-secure transports, such as TCP.

2.2.1. HTTP-over-Secure-Transport Request Type

When replying to an HTTP request that was conveyed over a secure transport, an Expect-CT Host SHOULD include in its response exactly one Expect-CT header field. The header field MUST satisfy the grammar specified in Section 2.1.

Establishing a given host as an Expect-CT Host, in the context of a given UA, is accomplished as follows:

1. Over the HTTP protocol running over secure transport, by correctly returning (per this specification) at least one valid Expect-CT header field to the UA.
2. Through other mechanisms, such as a client-side preloaded Expect-CT Host list.

2.2.2. HTTP Request Type

Expect-CT Hosts SHOULD NOT include the Expect-CT header field in HTTP responses conveyed over non-secure transport. UAs MUST ignore any Expect-CT header received in an HTTP response conveyed over non-secure transport.

2.3. User Agent Processing Model

The UA processing model relies on parsing domain names. Note that internationalized domain names SHALL be canonicalized according to the scheme in Section 10 of [RFC6797].

2.3.1. Expect-CT Header Field Processing

If the UA receives, over a secure transport, an HTTP response that includes an Expect-CT header field conforming to the grammar specified in Section 2.1, the UA MUST evaluate the connection on which the header was received for compliance with the UA's CT Policy, and then process the Expect-CT header field as follows.

If the connection complies with the UA's CT Policy (i.e. the connection is CT-qualified), then the UA MUST either:

- o Note the host as a Known Expect-CT Host if it is not already so noted (see Section 2.4), or
- o Update the UA's cached information for the Known Expect-CT Host if the "enforce", "max-age", or "report-uri" header field value directives convey information different from that already maintained by the UA. If the "max-age" directive has a value of 0, the UA MUST remove its cached Expect-CT information if the host was previously noted as a Known Expect-CT Host, and MUST NOT note this host as a Known Expect-CT Host if it is not already noted.

If the connection does not comply with the UA's CT Policy (i.e. is not CT-qualified), then the UA MUST NOT note this host as a Known Expect-CT Host.

If the header field includes a "report-uri" directive, and the connection does not comply with the UA's CT Policy (i.e. the connection is not CT-qualified), and the UA has not already sent an Expect-CT report for this connection, then the UA SHOULD send a report to the specified "report-uri" as specified in Section 3.

If a UA receives more than one Expect-CT header field in an HTTP response message over secure transport, then the UA MUST process only the first Expect-CT header field.

The UA MUST ignore any Expect-CT header field not conforming to the grammar specified in Section 2.1.

2.3.2. Noting an Expect-CT Host - Storage Model

The "effective Expect-CT date" of a Known Expect-CT Host is the time that the UA observed a valid Expect-CT header for the host. The "effective expiration date" of a Known Expect-CT Host is the effective Expect-CT date plus the max-age. An Expect-CT Host is "expired" if the effective expiration date refers to a date in the past. The UA MUST ignore any expired Expect-CT Hosts in its cache.

Known Expect-CT Hosts are identified only by domain names, and never IP addresses. If the substring matching the host production from the Request-URI (of the message to which the host responded) syntactically matches the IP-literal or IPv4address productions from Section 3.2.2 of [RFC3986], then the UA MUST NOT note this host as a Known Expect-CT Host.

Otherwise, if the substring does not congruently match an existing Known Expect-CT Host's domain name, per the matching procedure specified in Section 8.2 of [RFC6797], then the UA MUST add this host to the Known Expect-CT Host cache. The UA caches:

- o the Expect-CT Host's domain name,
- o whether the "enforce" directive is present
- o the effective expiration date, or enough information to calculate it (the effective Expect-CT date and the value of the "max-age" directive),
- o the value of the "report-uri" directive, if present.

If any other metadata from optional or future Expect-CT header directives are present in the Expect-CT header, and the UA understands them, the UA MAY note them as well.

UAs MAY set an upper limit on the value of max-age, so that UAs that have noted erroneous Expect-CT hosts (whether by accident or due to attack) have some chance of recovering over time. If the server sets a max-age greater than the UA's upper limit, the UA MAY behave as if the server set the max-age to the UA's upper limit. For example, if the UA caps max-age at 5,184,000 seconds (60 days), and a Pinned Host sets a max-age directive of 90 days in its Expect-CT header, the UA MAY behave as if the max-age were effectively 60 days. (One way to achieve this behavior is for the UA to simply store a value of 60 days instead of the 90-day value provided by the Expect-CT host.)

2.3.3. HTTP-Equiv <meta> Element Attribute

UAs MUST NOT heed "http-equiv="Expect-CT"" attribute settings on "<meta>" elements [W3C.REC-html401-19991224] in received content.

2.4. Noting Expect-CT

Upon receipt of the Expect-CT response header field, the UA notes the host as a Known Expect-CT Host, storing the host's domain name and its associated Expect-CT directives in non-volatile storage. The domain name and associated Expect-CT directives are collectively known as "Expect-CT metadata".

The UA MUST note a host as a Known Expect-CT Host if and only if it received the Expect-CT response header field over an error-free TLS connection, including the validation added in Section 2.5.

To note a host as a Known Expect-CT Host, the UA MUST set its Expect-CT metadata given in the most recently received valid Expect-CT header.

For forward compatibility, the UA MUST ignore any unrecognized Expect-CT header directives, while still processing those directives it does recognize. Section 2.1 specifies the directives "enforce", "max-age", and "report-uri", but future specifications and implementations might use additional directives.

2.5. Evaluating Expect-CT Connections for CT Compliance

When a UA connects to a Known Expect-CT Host using a TLS connection, if the TLS connection has errors, the UA MUST terminate the connection without allowing the user to proceed anyway. (This behavior is the same as that required by [RFC6797].)

If the connection has no errors, then the UA will apply an additional correctness check: compliance with a CT Policy. A UA should evaluate compliance with its CT Policy whenever connecting to a Known Expect-CT Host, as soon as possible. It is acceptable to skip this CT compliance check for some hosts according to local policy. For example, a UA may disable CT compliance checks for hosts whose validated certificate chain terminates at a user-defined trust anchor, rather than a trust anchor built-in to the UA (or underlying platform).

If a connection to a Known CT Host violates the UA's CT policy (i.e. the connection is not CT-qualified), and if the Known Expect-CT Host's Expect-CT metadata indicates an "enforce" configuration, the UA MUST treat the CT compliance failure as a non-recoverable error.

If a connection to a Known CT Host violates the UA's CT policy, and if the Known Expect-CT Host's Expect-CT metadata includes a "report-uri", the UA SHOULD send an Expect-CT report to that "report-uri" (Section 3).

A UA that has previously noted a host as a Known Expect-CT Host MUST evaluate CT compliance when setting up the TLS session, before beginning an HTTP conversation over the TLS channel.

If the UA does not evaluate CT compliance, e.g. because the user has elected to disable it, or because a presented certificate chain chains up to a user-defined trust anchor, UAs SHOULD NOT send Expect-CT reports.

3. Reporting Expect-CT Failure

When the UA attempts to connect to a Known Expect-CT Host and the connection is not CT-qualified, the UA SHOULD report Expect-CT failures to the "report-uri", if any, in the Known Expect-CT Host's Expect-CT metadata.

When the UA receives an Expect-CT response header field over a connection that is not CT-qualified, if the UA has not already sent an Expect-CT report for this connection, then the UA SHOULD report Expect-CT failures to the configured "report-uri", if any.

3.1. Generating a violation report

To generate a violation report object, the UA constructs a JSON message of the following form:

```
{
  "date-time": date-time,
  "hostname": hostname,
  "port": port,
  "effective-expiration-date": expiration-date,
  "served-certificate-chain": [ (MUST be in the order served)
    pem1, ... pemN
  ],
  "validated-certificate-chain": [
    pem1, ... pemN
  ],
  "scts": [
    sct1, ... sctN
  ]
}
```

Figure 4: JSON format of a violation report object

Whitespace outside of quoted strings is not significant. The key/value pairs may appear in any order, but each MUST appear only once.

The "date-time" indicates the time the UA observed the CT compliance failure. It is provided as a string formatted according to Section 5.6, "Internet Date/Time Format", of RFC 3339 [RFC3339].

The "hostname" is the hostname to which the UA made the original request that failed the CT compliance check. It is provided as a string.

The "port" is the port to which the UA made the original request that failed the CT compliance check. It is provided as an integer.

The "effective-expiration-date" is the Effective Expiration Date for the Expect-CT Host that failed the CT compliance check. It is provided as a string formatted according to Section 5.6, "Internet Date/Time Format", of RFC 3339 [RFC3339].

The "served-certificate-chain" is the certificate chain, as served by the Expect-CT Host during TLS session setup. It is provided as an array of strings, which MUST appear in the order that the certificates were served; each string "pem1", ... "pemN" is the Privacy-Enhanced Mail (PEM) representation of each X.509 certificate as described in RFC 7468 [RFC7468].

The "validated-certificate-chain" is the certificate chain, as constructed by the UA during certificate chain verification. (This may differ from the "served-certificate-chain".) It is provided as an array of strings, which MUST appear in the order matching the chain that the UA validated; each string "pem1", ... "pemN" is the Privacy-Enhanced Mail (PEM) representation of each X.509 certificate as described in RFC 7468 [RFC7468].

The "scts" are JSON messages representing the SCTs (if any) that the UA received for the Expect-CT host and their validation statuses. The format of "sct1", ... "sctN" is shown in Figure 5. The SCTs may appear in any order.

```
{
  "sct": sct,
  "status": status,
  "source": source
}
```

Figure 5: JSON format of an SCT object

The "sct" is as defined in Section 4.1 of RFC 6962 [RFC6962].

The "status" is a string that the UA MUST set to one of the following values: "unknown" (indicating that the UA does not have or does not trust the public key of the log from which the SCT was issued), "valid" (indicating that the UA successfully validated the SCT as described in Section 5.2 of [RFC6962]), or "invalid" (indicating that the SCT validation failed because of, e.g., a bad signature).

The "source" is a string that indicates from where the UA obtained the SCT, as defined in Section 3.3 of RFC 6962 [RFC6962]. The UA MUST set "source" to one of the following values: "tls-extension", "ocsp", or "embedded".

3.2. Sending a violation report

When an Expect-CT header field contains the "report-uri" directive, and the connection does not comply with the UA's CT Policy, or when the UA connects to a Known Expect-CT Host with Expect-CT metadata that contains a "report-uri", the UA SHOULD report the failure as follows:

1. Prepare a JSON object "report object" with the single key "expect-ct-report", whose value is the result of generating a violation report object as described in Figure 4.
2. Let "report body" be the JSON stringification of "report object".
3. Let "report-uri" be the value of the "report-uri" directive in the Expect-CT header field.
4. Queue a task [1] to fetch [2] "report-uri", with the synchronous flag not set, using HTTP method "POST", with a "Content-Type" header field of "application/expect-ct-report", and an entity body consisting of "report body".

4. Security Considerations

4.1. Maximum max-age

1 year?

5. Privacy Considerations

6. IANA Considerations

7. Usability Considerations

When the UA detects a Known Expect-CT Host in violation of the UA's CT Policy, users will experience denials of service. It is advisable for UAs to explain the reason why.

It is advisable that UAs have a way for users to clear Known Expect-CT Hosts and that UAs allow users to query Known Expect-CT Hosts.

8. References

8.1. 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>>.
- [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>>.
- [RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <<http://www.rfc-editor.org/info/rfc3339>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <<http://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, <<http://www.rfc-editor.org/info/rfc7230>>.
- [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>>.

[RFC7468] Josefsson, S. and S. Leonard, "Textual Encodings of PKIX, PKCS, and CMS Structures", RFC 7468, DOI 10.17487/RFC7468, April 2015, <<http://www.rfc-editor.org/info/rfc7468>>.

[W3C.REC-html401-19991224]
Raggett, D., Hors, A., and I. Jacobs, "HTML 4.01 Specification", World Wide Web Consortium Recommendation REC-html401-19991224, December 1999, <<http://www.w3.org/TR/1999/REC-html401-19991224>>.

8.2. URIs

[1] <https://html.spec.whatwg.org/#queue-a-task>

[2] <https://fetch.spec.whatwg.org/#fetching>

Author's Address

Emily Stark
Google

Email: estark@google.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.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

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 implementations 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 underivable, 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., Dukkupati, 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-dukkupati-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: Informational
Expires: April 21, 2017

I. Svirid
October 18, 2016

WebSocket2 over HTTP/2
draft-svirid-websocket2-over-http2-00

Abstract

This document specifies a new protocol called WebSocket2 on top of HTTP/2. The WebSocket2 protocol enables two-way binary communication between a client running untrusted code in a controlled environment to a remote host that has opted-in to communications from that code.

This protocol has little in common with the WebSocket protocol [RFC6455] other than client side API compatibility.

Please send feedback to the ietf-http-wg@w3.org mailing list.

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 21, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Conventions and Terminology	3
2. Overview	3
2.1. WebSocket2 Protocol	3
3. Handshake	3
3.1. Client Handshake Request	3
3.2. Server Handshake Reply	4
3.2.1. Handshake Error Reasons	5
3.3. Post Handshake	5
4. Data Framing	5
4.1. Text Frame	6
4.2. Binary Frame	6
4.3. Error Frame	6
4.3.1. Error Frame Codes	7
5. VarSize	8
6. Compression	8
6.1. LZ4 Compressed Payload	8
6.2. Deflate Compressed Payload	8
7. References	9
7.1. Normative References	9
7.2. Informative References	9
Appendix A. Acknowledgements	9
Author's Address	9

1. Introduction

You can read about why two way data streaming is important from the introduction to WebSockets in RFC6455 Section 1
<<https://tools.ietf.org/html/rfc6455#section-1>>.

WebSocket2 over HTTP/2 is a protocol on top HTTP/2 designed for modern times. Previous WebSocket client side API will be fully backward compatible with WebSocket2.

In this document, we describe WebSocket2 and how to layer WebSocket2 semantics onto HTTP/2 semantics by defining detailed mapping, replacement of operations and events.

1.1. Conventions and Terminology

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 RFC 2119 [RFC2119].

2. Overview

WebSocket2 is functionally equivalent to binary streaming between a sandboxed client and inexplicit host, but layered on top of HTTP2.

Key advantages of WebSocket2 over WebSocket:

- o No masking
- o Simpler handshake and negotiation
- o Lz4 compression method

Key advantages of WebSocket2 over HTTP/2 include:

- o Two-way real time communication
- o Server push and delivery without client request
- o Binary transmission

2.1. WebSocket2 Protocol

The protocol has two main parts, the handshake and data transfer. Data transmitted using WebSocket2 supports compression.

3. Handshake

The main job of the handshake is to request authority and if granted by the server, to negotiate a compression medium.

3.1. Client Handshake Request

The client MUST use the :method GET.

The client MUST send a sec-ws2-version header that MUST specify the websocket2 version being used.

The client MAY send a sec-ws2-compression header that advertises the compression methods the client supports. Valid key value pairs include:

- o lz4=1-9;
 - * This client supports lz4 with compression levels from 1 to 9
- o lz4=1;
 - * This client supports lz4 with compression levels 1
- o deflate=8-15;
 - * This client supports deflate with sliding window bits from 8-15

Duplicate keys MUST NOT be present.

The client MUST NOT set the END_STREAM flag when sending the headers.

A client handshake request may look like:

```
:method: GET
:scheme: wss
:authority: example.org
:path: /demo
sec-ws2-version: 1
sec-ws2-compression: lz4=1-9; deflate=8-15;
```

3.2. Server Handshake Reply

END_STREAM on the HTTP/2 Header frame MUST only be set in the case of rejection.

The server MUST send ONLY ONE of the advertised compression methods or exclude the sec-ws2-compression header from the response, signaling that no compression will be used.

The server MUST include the sec-ws2-error header in the reply with an outlined error reason.

A successful server handshake reply may look like:

```
:status: 200
sec-ws2-compression: lz4=1;
sec-ws2-error: success
```

This signals that the server chose to use lz4 with a compression level of 1. Now both the client and server MUST use only this compression method.

3.2.1. Handshake Error Reasons

Valid error reasons are:

success

* The server accepted the client

invalid_version

* This version of websockets is not supported by the server

cannot_negotiate_compression

* This means the client did not offer any compression that the server requires

rejected

* This means the server rejected the client and does not want to say why. This means the server supports websockets, but rejected this particular client

3.3. Post Handshake

Following the handshake the client or server MUST NEVER set the END_STREAM flag on any HTTP/2 DATA frame UNLESS the stream is to be gracefully terminated. Only a HTTP/2 DATA frame containing a WebSocket2 error frame allow the END_STREAM flag to be set.

4. Data Framing

Once a handshake has been successfully completed the remote endpoints can begin to send data to each other. Data is sent using the HTTP/2 transport layer fully adhering to DATA Frames, Section 6.1 [RFC7540]. WebSocket2 has its own encapsulated framing protocol that is not to be confused with HTTP/2 DATA Frames.

Three frame types are defined:

text (represented by 0)

binary (represented by 1)

error (represented by 2)

Three compression types are defined:

none (represented by 0)

lz4 (represented by 1)

deflate (represented by 2)

A WebSocket2 over HTTP/2 frame starts with the full frame length in a special format called VarSize. If the first octet is less than 254,

this is the full frame length. If the first octet is 254, read the next 16 bits as a little endian unsigned number for the frame length. If the first octet is 255, read the next 32 bits as a little endian unsigned number for the frame length.

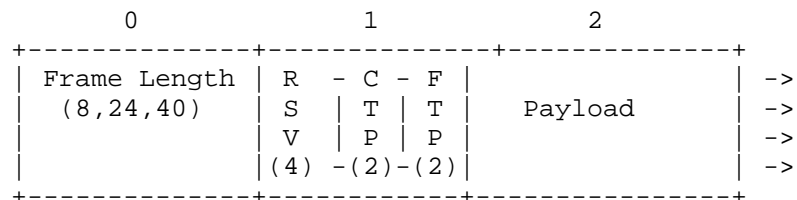
Next are 4 reserved bits that MUST be set to 0.

The next 2 bits specify the compression type.

The next 2 bits specify the frame type.

Next is the payload which MAY be compressed if compression was negotiated.

The term PAYLOAD in this section refers to data AFTER decompression if compression was negotiated.



4.1. Text Frame

The text frame has a Frame Type value of 0. The payload must be UTF-8 encoded text. The whole message MUST contain valid UTF-8. Invalid UTF-8 in the payload requires the remote end point to send an error frame and close their side of the stream.

4.2. Binary Frame

The binary frame has a Frame Type value of 1. The payload must be arbitrary binary data which the application layer passes without comprehension.

4.3. Error Frame

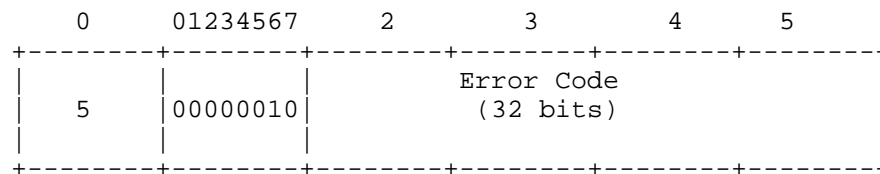
The error frame has a Frame Type value of 2. It MUST contain a VALID 32 bit error code. If the code is shorter than 32 bits, 0 bits MUST make up the rest to fill a total of 32 bits. Currently there are no error codes less than 32 bits.

The HTTP/2 transport layer DATA frame carrying the WebSocket2 error frame MUST have the END_STREAM flag set.

No further WebSocket2 frames may be sent from this point onward and the stream is half closed.

The remote endpoint that receives the error frame MUST flush all pending sends followed by an error frame of its own with the CLOS error code. The HTTP/2 Data frame carrying this WebSocket2 frame MUST have the END_STREAM flag set.

The full WebSocket2 error frame with the length:



4.3.1. Error Frame Codes

Valid error frame codes currently are:

CLOS

- * This should be sent when a client or server want to close the stream gracefully.

UTF8

- * This should be sent when invalid UTF8 was passed in a text frame by a remote endpoint.

DECO

- * This should be sent when decompression failed by a remote endpoint.

FRAM

- * This should be sent when an invalid Frame Type was specified.

LRGE

- * This should be sent when an endpoint rejects a frame due to it being too large. This is up to the endpoint.

5. VarSize

VarSize is a variable sized binary ranging from 1-5 octets. It represents an unsigned value. If the first octet is equal to 254, read the next 16 bits as little endian unsigned. If the first octet is equal to 255, read the next 32 bits as little endian unsigned. Otherwise if the first octet is less than 254, treat this as the final value.

6. Compression

WebSocket defined one compression method which used deflate and kept a sliding window. This compression is great but has limitations. Also keeping a sliding window is memory intensive.

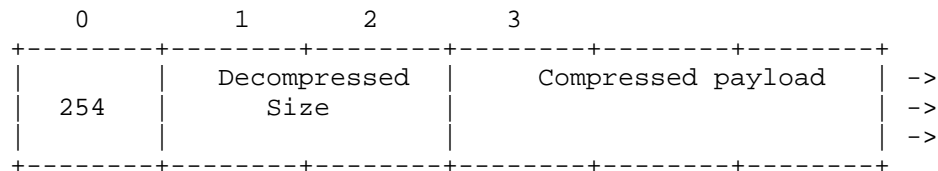
Lz4 is a compression method that is great for any kind of data while being very cheap.

Currently two compression methods are defined lz4 and deflate. The protocol is open to tweaking or accepting more in the future.

6.1. LZ4 Compressed Payload

A lz4 compressed payload is a VarSize number of the decompressed size followed by the actual compressed payload. The lz4 compression level to use MUST be what was negotiated in the handshake.

A lz4 compressed payload may look like:



6.2. Deflate Compressed Payload

The deflate compression method implements [RFC7692] but defines more strict bounds. There is no ability to reset compression context.

- * Both client and server MUST use the sliding window bits as determined by the server.
- * The client MUST use a memory level of 8.
- * The client MUST use a compression level of 9 (best_compression).

Any sliding window MUST NEVER have its context reset.

If the deflated payload trailing 4 octets are 0x00, 0x00, 0xFF, 0xFF, remove them before sending the payload.

Before inflating the payload append 0x00, 0x00, 0xFF, 0xFF to the end of it.

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>>.
- [RFC7692] Yoshino, T., "Compression Extensions for WebSocket", RFC 7692, DOI 10.17487/RFC7692, December 2015, <<http://www.rfc-editor.org/info/rfc7692>>.

7.2. Informative References

- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<http://www.rfc-editor.org/info/rfc6455>>.
- [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>>.

Appendix A. Acknowledgements

The author wishes to thank Kari hurttta for contributing the handshake.

The author wishes to thank the participants of the WebSocket protocol, participants of the HTTP/2 protocol and participants of the QUIC protocol.

Author's Address

Ivan Svirid
Toronto, ON
Canada

Email: vans_163@yahoo.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 6, 2018

V. Krasnov
Cloudflare, Inc.
Y. Weiss
Akamai Technologies, Inc.
March 5, 2018

Compression Dictionaries for HTTP/2
draft-vkrasnov-h2-compression-dictionaries-03

Abstract

This document specifies new HTTP/2 frame types and new HTTP/2 settings values that enable the use of previously transferred data as compression dictionaries, significantly improving overall compression ratio for a given connection.

In addition, this document proposes to define a set of industry standard, static, dictionaries to be used with any Lempel-Ziv based compression for the common textual MIME types prevalent on the web.

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 September 6, 2018.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
1.1. Conventions and Terminology	3
2. Preliminaries	3
2.1. Security Considerations	3
2.2. Content Coding	3
2.3. Compression Contexts	4
2.4. Server Push Interaction	4
2.5. HTTP/QUIC	4
3. HTTP/2 Extension	4
3.1. Extension Settings	4
3.2. Extension Frames	5
3.2.1. The SET_COMPRESSION_CONTEXT frame	5
3.2.2. The SET_DICTIONARY Frame	5
3.2.3. The USE_DICTIONARY Frame	7
3.3. Static Dictionaries	7
4. Dictionary State	8
4.1. Attack scenarios and mitigations	10
4.1.1. Cross-origin secret leak	10
4.1.2. Same-origin secret leak	11
5. References	11
5.1. Normative References	12
5.2. Informative References	12
Authors' Addresses	12

1. Introduction

The HTTP/2 [RFC7540] protocol encourages the use of many small assets for CSS/JS/HTML, due to its multiplexed nature. Prior to HTTP/2, asset inlining was encouraged, resulting in fewer, larger assets per website.

The HTTP/2 protocol also allows for transmitted data to be compressed with a lossless compression format. The format used is specified in the "Content-Encoding" (see [RFC2616], section 14.11) header field. For example, "Content-Encoding: br" means the data was compressed using the Brotli format.

The nature of the compression algorithms, such as DEFLATE [RFC1951] and Brotli [RFC7932], used with HTTP in practice, require a certain "window" of data to perform backward matching. Therefore, larger files have much better compression ratio. To improve compression for

smaller files, these algorithms allow to use a chunk of arbitrary data as a "Custom Dictionary" and function as the initial sliding window.

Note: While that is not longer true for the latest stable version of Brotli, there's work underway to re-enable use of arbitrary compression dictionaries.

Compression is a compute-heavy operation, where investing additional compute power results in diminishing returns (in terms of compression ratio/CPU cycles). The "Custom Dictionary" technique is known to improve compression ratio significantly, with little additional computational cost. It is also supported by most Lempel-Ziv based compression formats.

This document introduces a mechanism for using previously transmitted data over HTTP/2 as a dictionary to be used with an underlying compression algorithm.

1.1. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Preliminaries

2.1. Security Considerations

The use of compression over an encrypted connection could be used by malicious actors to potentially leak sensitive information. We will collaborate with industry experts to identify any additional attack vectors introduced by this draft, and include a set of best practices to both servers and clients that would implement it.

A list of attack vectors and potential mitigations is described later in this document.

2.2. Content Coding

A server that wishes to apply protocol level compression on a stream or use a stream as a dictionary SHOULD not apply non-identity content-coding (see [RFC7231], section 3.1.2.1) to that stream.

2.3. Compression Contexts

In the scope of this document, a compression context is a set of non-overlapping streams, that SHALL only be used as compression dictionaries for streams within the same compression context. While it is the responsibility of the server to implement best-practice techniques to mitigate cross-compression side channel attacks, compression contexts let the client mitigate some of the risks of cross-compression side channel attacks, by explicitly stating which requests can be cross-compressed with which requests.

For example a client may choose to disable compression for cross-site requests by assigning them to different compression contexts.

2.4. Server Push Interaction

Pushed streams may be cross-stream compressed or used as dictionaries, same as a regular stream. In some scenarios it may benefit the server to push a dummy resource to prime a dictionary.

2.5. HTTP/QUIC

Due to the nature of this draft, it is expected that a strict order is maintained between the definition and consumption of dictionaries. The nature of QUIC is such that frames and streams might not be delivered in the order they are sent, therefore, a head-of-line blocking may occur when implementing compression dictionaries in HTTP/QUIC. This is similar to the tradeoff present in the HPACK/QUIC mapping.

3. HTTP/2 Extension

3.1. Extension Settings

The extension introduces a new SETTINGS value.

SETTINGS_COMPRESSION(0xTBA): For greater compression, and to prevent setting identifier depletion, the 32-bit value for this setting is defined as follows:

```
+-----+-----+-----+-----+
| SDVersion (8) | Fmt (8) | DSize (8) | NDict (8) |
+-----+-----+-----+-----+
```

NDict: Indicates the number of dictionaries the client is willing to maintain. The default value is 0, the maximal value is 255.

DSize: Log2 of the maximal size of each dictionary. The default value is 0, the maximal value is 255. For example value of 17 indicates each dictionary MUST be smaller or equal to 2^{17} (131,072 octets).

Fmt: Compression format to use, as a bitmask. 1st bit indicates brotli, 2nd bit indicates zlib. Other bits are reserved for future compression methods. A value of 0 indicates no support for cross-stream compression.

SDVersion: If greater than 0, indicates the version of static dictionaries to use. Maximal value is 255, the default value is 0, which indicates no static dictionaries are used.

3.2. Extension Frames

3.2.1. The SET_COMPRESSION_CONTEXT frame

The SET_COMPRESSION_CONTEXT frame (type=0xTBA).

```
+-----+
| Context (8) |
+-----+
```

The SET_COMPRESSION_CONTEXT frame can be sent by the client on any stream in the idle state. The frame indicates the compression context ID for the given stream. Frames with an assigned context SHALL NOT be compressed using dictionaries from a different context. Frames with an assigned context SHALL NOT be used as a dictionary for streams with from a different context.

The SET_COMPRESSION_CONTEXT frame contains the following fields:

Context: an 8-bit context ID that indicates the compression context for the stream. If the frame is omitted, then the context value is assumed to be 0. The allowed context values are 0 through 255. A special context ID of 255 indicates the stream can only be compressed using the static dictionaries.

3.2.2. The SET_DICTIONARY Frame

The SET_DICTIONARY frame (type=0xTBA) contains one to many Dictionary-Entry.

```
+-----+-----+
| Dictionary-Entry (+) ... |
+-----+-----+
```

A Dictionary-Entry field is encoded as follows:

```
+-----+
|      Dictionary-ID (8)      |
+-----+-----+
| P |      Size (7+)      |
+-----+-----+
| E? | D? | Truncate? (6+) |
+-----+-----+
|      Offset? (8+)      |
+-----+
```

The SET_DICTIONARY frame can be sent from the server to the client, on any client initiated stream in the open or half-closed (remote) states, or on any server initiated stream in the reserved (local) state. The SET_DICTIONARY frame MUST precede any DATA frames on that stream. The SET_DICTIONARY frame SHOULD be followed by sufficient DATA frames to build the dictionaries. If a RST frame was received for the stream before sufficient DATA was sent, the dictionaries are reset.

The Dictionary-Entry contains the following fields:

Dictionary-ID: an 8-bit ID, indicates the dictionary. MUST be lower than the value agreed by the SETTINGS_COMPRESSION setting.

Size: Indicates how many octets of the stream will be used for the dictionary. Size is represented as an integer with 7-bit prefix (see [RFC7541], Section 5.1). If P is set, the actual number of octets to use is 2 to the power of Size. If the computed value is greater than the length of the decompressed DATA, use all the available DATA.

Truncate: An optional field, represented as an integer with 6-bit prefix. Present when the APPEND flag is set. Truncate indicates the number of octets to keep of the existing dictionary, before appending the new data to it. If E is set, then Truncate is ignored, and new data is appended at the end. If Truncate is zero, then the dictionary is replaced, as if APPEND was unset. If the optional field D is set, then the first Truncate octets of the previous dictionary are used, otherwise the last Truncate octets are used.

Offset: An optional field, represented as an integer with 8-bit prefix. Present when the OFFSET flag is set. Offset indicates that the first Offset octets of the stream are ignored when building the dictionary.

The flags defined for the SET_DICTIONARY frame apply to each Dictionary-Entry in the frame. The SET_DICTIONARY frame defines the following flags:

APPEND (0x1): Indicates that the data is to be appended to the existing dictionary with the given ID, as opposed to replacing it with the new data. Also indicates that fields E, D and Truncate are present.

OFFSET (0x2): Indicates the presence of the Offset field.

3.2.3. The USE_DICTIONARY Frame

The USE_DICTIONARY frame (type=0xTBA).

```
+-----+
| Dict ID (8) |
+-----+
```

The USE_DICTIONARY frame indicates that the current stream is compressed with the indicated dictionary. The USE_DICTIONARY frame MUST be sent prior to any DATA frame on a given stream. SET_DICTIONARY and USE_DICTIONARY frames MAY be sent on the same stream. Only one USE_DICTIONARY frame MAY be sent for a stream.

The USE_DICTIONARY frame contains the following fields:

Dict ID: an 8-bit ID that indicates which dictionary to use. The dictionary MUST be previously defined by a SET_DICTIONARY frame, or by a static dictionary.

3.3. Static Dictionaries

This document proposes to generate a set of up to 8 standard dictionaries to be optionally bundled with supporting implementations. Each dictionary should be 32,768 or 65,536 octets long.

Each static dictionary will be identified by an integer ID in the range {0..7}.

If either endpoint supports the use of static dictionaries, it will indicate this by setting the SDVersion value of SETTINGS_COMPRESSION to greater than 0. The number will indicate the highest version of the dictionaries known.

The actual version used will be the lowest of the two values set by the endpoints.

If the client and the server agree on the use of static dictionaries, then both will initialize the first 8 dictionaries (IDs 0 through 7), with the contents of the static dictionaries. The static dictionaries belong to context 0.

If the value of the field NDict is lower than 8, then up to NDict dictionaries will be initialized.

4. Dictionary State

Both the server and the client MUST process the SET_DICTIONARY and USE_DICTIONARY frames in the order they are sent/received, with the exception when both are sent over the same stream. In that case USE_DICTIONARY is processed prior to the SET_DICTIONARY frames.

Doing otherwise will result in an illegal state of the dictionaries. This is similar to the way HEADER frames are processed in order to maintain legal HPACK state on the server and the client.

A possible dictionary implementation can be describes as follows:

```
struct {
    u8  id;
    u8  ctx;
    u64 size;
    u8  dict[size];
} D;
```

The collection of dictionaries could then be described as:

```
D dictionaries[NDict];
```

Initially all the dictionaries are uninitialized:

```
for (i = 0; i < NDict; i++) {
    dictionaries[i] = {id = i, ctx = 0, size = 0, dict = {}};
}
```

Client side USE_DICTIONARY frame behaviour pseudo code:

```
dictionary = dictionaries[frame.Dictionary-ID]
```

```
if (dictionary.ctx != 0 && dictionary.ctx != stream.ctx)
    return PROTOCOL_ERROR
```

```
stream.decompressed_data = decompress(stream.dict, stream.data)
```

Client side SET_DICTIONARY frame behaviour pseudo code:

```
foreach entry = frame.Dictionary-Entry {
    dictionary = dictionaries[entry.DICT_ID]

    if (entry.size == 0) {
        dictionary.size = 0
        dictionary.ctx = 0
        dictionary.dict = {}
        continue
    }

    if (dictionary.ctx != 0 && dictionary.ctx != stream.ctx) {
        return PROTOCOL_ERROR
    }

    dictionary.ctx = stream.ctx

    if (entry.P == 1) {
        size = 1 << entry.Size
    } else {
        size = entry.Size
    }

    if (frame.APPEND) {
        if (entry.E == 1) {
            truncate = dictionary.size
        } else {
            truncate = entry.Truncate
        }
    } else {
        truncate = 0
    }

    if (frame.OFFSET) {
        offset = entry.Offset
    } else {
        offset = 0
    }

    new_dict_data = stream.decompressed_data[offset:offset + size]
    if (entry.D == 1) {
        old_dict_data = head(dictionary.dict, truncate)
    } else {
        old_dict_data = tail(dictionary.dict, truncate)
    }

    dict_data = append(old_dict_data, new_dict_data)

    dictionary.dict = tail(dict_data, 1 << settings.DSize)
```

```
    dictionary.size = len(dictionary.dict)
}
```

The server behaviour mirrors the client behaviour, but it is up to the server to choose the best dictionary.

4.1. Attack scenarios and mitigations

A single HTTP/2 connection is likely to be shared among multiple origins (over which it is authoritative) and among different navigation contexts to the same origin. When such sharing happens, and if compression contexts are shared between those instances, an attacker can use a BREACH-style attack in order to exfiltrate secrets from the context. Such secrets may include:

- o Cookies set using Javascript (and in-particular "httponly" cookies set from anonymous functions in external JS, which is not accessible to scripts otherwise)
- o CSRF tokens
- o CSP nonces
- o Application level secrets (e.g. financial information, stored credit cards numbers, codes, etc.)

The mechanism for such data theft can happen if the attacker can: *

- * Download multiple similar payloads to the target page modulo the actual secret, while trying out multiple permutations of the secret.
- * Observe the on-the-wire transfer size using Resource Timing's "transferSize" property.

The rest of this section will describe different scenarios where those conditions are met as well as potential mitigations for them.

4.1.1. Cross-origin secret leak

An HTTP/2 session can be used to deliver resources from multiple origins over which the session has proved to be authoritative, through connection reuse (see [RFC7540] section 9.1.1 for more details). As a result, sharing compression contexts between such origins can be theoretically used to leak secrets from one of these origins to the next.

4.1.1.1. Mitigation

Limiting compression contexts to be used within the confines of a single origin.

4.1.2. Same-origin secret leak

Malicious pages on the origin as well as an XSS attacker can normally use "fetch()" or "XMLHttpRequest()" in order to inspect in-content secrets. This could be limited with CSP by only permitting the download of specific files, using nonces or using "connect-src 'none'" in order to limit arbitrary scripts from downloading files that contain secrets. However, using shared-dictionaries between secret resources and malicious ones can enable an attacker to guess said secrets and exfiltrate them (e.g. using other deficiencies in the defined CSP, if there are any).

Furthermore, said malicious page or XSS attack can also use as a dictionary resources fetched from the same origin in a different browsing context, enabling it to also inspect resources which cannot be fetched at all on its base page.

4.1.2.1. Mitigation

There's no obvious mitigation for this kind of attack, but a few options are:

- o Limiting compression contexts to be used only within a single navigation context can limit the opportunity for the separate navigation context to inspect secrets from resources it is not allowed to fetch. At the same time this can be complex to implement, as the network layer is not aware of the navigation context and is supposed for example to dedupe outgoing requests from different compression contexts.
- o "transferSize" padding/bucketing in such cases (e.g. pages with above mentioned CSP limitations) may be enough to render this attack not-practical.
- o Limit dictionary sharing (or "transferSize" accuracy for resources that use shared dictionaries) only to non-credentialed resource fetches.

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>>.
- [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, <<https://www.rfc-editor.org/info/rfc2616>>.
- [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>>.
- [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>>.
- [RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <<https://www.rfc-editor.org/info/rfc7541>>.

5.2. Informative References

- [BREACH] Prado, A., Harris, N., and Y. Gluck, "BREACH: SSL, Gone in 30 Seconds", 2013, <<http://breachattack.com/>>.
- [RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, DOI 10.17487/RFC1951, May 1996, <<https://www.rfc-editor.org/info/rfc1951>>.
- [RFC7932] Alakuijala, J. and Z. Szabadka, "Brotli Compressed Data Format", RFC 7932, DOI 10.17487/RFC7932, July 2016, <<https://www.rfc-editor.org/info/rfc7932>>.

Authors' Addresses

Vlad Krasnov
Cloudflare, Inc.

Email: vlad@cloudflare.com

Yoav Weiss
Akamai Technologies, Inc.

Email: yoav@yoav.ws

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 11, 2017

T. Yoshino
W. Zhu
Google, Inc.
May 10, 2017

WiSH: A General Purpose Message Framing over Byte-Stream Oriented Wire
Protocols (HTTP)
draft-yoshino-wish-03

Abstract

This document defines a general purpose message framing named WiSH which supports bi-directional (bidi) message-based communication over byte-stream oriented protocols such as HTTP (in its standard semantics). WiSH is designed to be compatible with WebSocket. WiSH can be viewed as a binary and bidi alternative to the framing defined for the server-sent events (SSE, EventSource) Web API [SSE].

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 11, 2017.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Background	3
3. Conformance Requirements and Terminology	3
4. WiSH Protocol	4
5. Framing	4
6. Using WiSH over HTTP	5
7. WebSocket Compatibility Consideration	5
7.1. Valid UTF-8 Requirement	5
8. Acknowledgements	6
9. References	6
9.1. Normative References	6
9.2. Non-normative References	6
Authors' Addresses	7

1. Introduction

The WebSocket protocol was proposed to provide native client-server bidi messaging for the Web. It has been implemented and deployed widely, but there are still missing semantics and features. See [BidiwebSurvey].

WiSH is a general purpose message framing for use over the standard HTTP semantics to provide bidi messaging semantics. WiSH stands for Web in Strict HTTP.

The communication protocol providing the standard HTTP semantics can be HTTP/1.1 [RFC7231], HTTP/2 [RFC7540], HTTP/2 + QUIC [QUIC], or any future protocols. Wire protocol features such as multiplexing, session priority, etc. are provided by the underlying protocol [TransportAbstraction]. Unlike HTTP/2, HTTP/1.1 doesn't specify if earlier 2xx responses are allowed [RFC7540]. Therefore, when HTTP/1.1 is used as the underlying protocol, full-duplex communication may be broken if the client, server or any intermediary chooses to buffer or reject earlier 2xx responses. Since intermediaries may buffer response bodies, bidi communication over WiSH may experience extra latency compared to WebSocket. When HTTPS is used, response body buffering by intermediaries is less likely to happen.

The wire protocol features of WebSocket, such as handshake or control messages, are all dropped. WiSH respects the semantics of the underlying protocol (as opposed to turning it to a transport protocol). The concept of fragmentation is retained for enabling

starting message transmission before determining the final length of the message.

Application-level protocols may use WiSH as the framing protocol to support bidi communication over HTTP and for Web and Internet clients.

2. Background

There has been several attempts to improve bidi message-based communication on the Web.

The server-sent events (SSE) [SSE] realized message-based communication in the server-to-client direction, by introducing a new Web API and a special message framing format while using HTTP as the wire protocol. Except for the issue of possible buffering by intermediaries, SSE works well with existing intermediaries and frameworks that support HTTP.

WebSocket realized bidi message-based communication by introducing both a new Web API and a new wire protocol. Because the wire protocol is incompatible with HTTP, intermediaries and frameworks have to be upgraded to understand the wire protocol to support WebSocket.

In parallel to the development of WebSocket, HTTP has been greatly improved with HTTP/2. There are more improvements upcoming to the HTTP e.g. QUIC.

It's desirable that normal HTTP traffic and bidi message-based communication on the Web share further evolution to reduce cost of development and standardization. Bidi message-based communication on the Web should be multiplexed with normal HTTP traffic and should benefit from future transport-level improvements such as QUIC.

WiSH is designed based on the above analysis. Use of the standard HTTP semantics as-is reduces cost and makes the Web simpler.

3. Conformance Requirements and Terminology

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].

Requirements phrased in the imperative as part of algorithms (such as "strip any leading space characters" or "return false and abort these steps") are to be interpreted with the meaning of the key word ("MUST", "SHOULD", "MAY", etc.) used in introducing the algorithm.

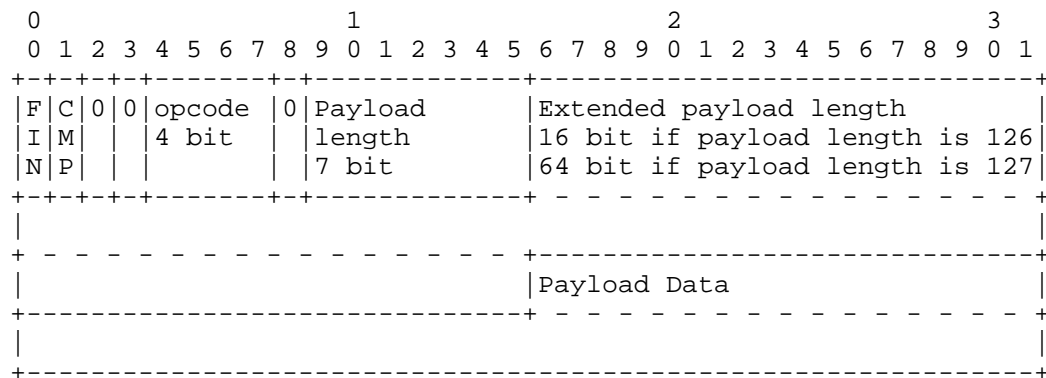
Conformance requirements phrased as algorithms or specific steps can be implemented in any manner, so long as the end result is equivalent. In particular, the algorithms defined in this specification are intended to be easy to understand and are not intended to be performant.

4. WiSH Protocol

WiSH frames messages over an HTTP request body or response body using the framing defined in Section 5.

The "Content-Type" header value of the underlying HTTP request or response message for which WiSH is used MUST be "application/web-stream".

5. Framing



WiSH is compatible with the framing of the WebSocket protocol [RFC6455].

The opcode field indicates how to interpret the payload data field. WiSH uses the following opcodes.

- o %x0 denotes a continuation frame
- o %x1 denotes the initial frame of a text message
- o %x2 denotes the initial frame of a binary message
- o %x3 denotes the initial frame of a text metadata message
- o %x4 denotes the initial frame of a binary metadata message

Any values not listed here are reserved.

The FIN bit together with the continuation frame opcode, payload length and extended payload length work in the same way as the WebSocket protocol to represent messages. The fragmentation mechanism allows for flushing part of a large message payload without waiting for the total size of the message to be determined.

The CMP bit indicates whether the message is compressed. The CMP bit of the first frame MUST be set to 1 when compression is enabled for the message. Otherwise, it MUST be set to 0. The CMP bit of non-first frames MUST be always set to 0.

The message type distinction by the opcode field (text and binary) is kept to allow better Web support.

The two metadata opcodes can be used for exchanging metadata e.g. using messages with opcode set to %x3 with metadata encoded in JSON in its payload field.

The status code and status reason defined in the WebSocket protocol are dropped.

The ping and pong control message of the WebSocket protocol are dropped. If such a feature is needed, it should be provided by underlying protocols.

The permessage-deflate extension [RFC7692] is defined for the WebSocket protocol, to add a compression mechanism to it. The permessage-deflate extension can be applied to WiSH. The details are to be specified.

What contents are exchanged and in what encoding they are exchanged over WiSH are to be defined by the application layer.

6. Using WiSH over HTTP

The standard HTTP (REST) semantics should be followed, especially the choice of the HTTP method. Some HTTP semantics may not be applicable, e.g. the "Cache-Control" header, when the body is streamed. However, such limitation is not specific to WiSH.

7. WebSocket Compatibility Consideration

7.1. Valid UTF-8 Requirement

In RFC6455, endpoints are required to Fail the WebSocket Connection when they find that the byte stream in a text message is not a valid UTF-8 stream. To conform to the requirement, RFC6455 server frameworks check UTF-8 validness. The contents of text messages of

WiSH also MUST be a valid UTF-8 stream. However, WiSH endpoints are not required to check UTF-8 validness. This provides more flexibility to server development. For example, a server may choose to check UTF-8 validness inside a JSON parser.

8. Acknowledgements

Thank you to the following people for giving feedback to the document: Ben Christensen, Costin Manolache, Kari Hurtta, Loic Huguin, Roberto Peon, Van Catha.

9. References

9.1. 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>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<http://www.rfc-editor.org/info/rfc6455>>.
- [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>>.
- [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>>.
- [RFC7692] Yoshino, T., "Compression Extensions for WebSocket", RFC 7692, DOI 10.17487/RFC7692, December 2015, <<http://www.rfc-editor.org/info/rfc7692>>.

9.2. Non-normative References

- [SSE] WHATWG, "HTML Living Standard - Server-sent events", May 2017, <<https://html.spec.whatwg.org/multipage/comms.html#server-sent-events>>.

[BidiwebSurvey]

Yoshino, T. and W. Zhu, "Non Request-Response Communication over the Web, and What's Missing", January 2014, <<https://github.com/bidiweb/bidiweb-semantics/blob/master/SurveyOfProtocolGaps.md>>.

[TransportAbstraction]

Zhu, W., "http-transport-abstraction", July 2016, <<https://github.com/bidiweb/http-transport-abstraction>>.

[QUIC]

Hamilton, R., Iyengar, J., Swett, I., and A. Wilk, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2", July 2016.

Authors' Addresses

Takeshi Yoshino
Google, Inc.

Email: tyoshino@google.com

Wenbo Zhu
Google, Inc.

Email: wenboz@google.com