HTTPbis
Internet-Draft
Intended status: Standards Track
Expires: November 25, 2018

M. Bishop
Akamai

May 24, 2018

The "SNI" Alt-Svc Parameter
draft-bishop-httpbis-sni-altsvc-02

Abstract

HTTP Alternative Services provides a mechanism for an origin to declare that its content is accessible via some other combination of host, port, and protocol. In the process of using such an alternative, an observer can identify that the client is requesting resources from a particular hostname.

This document extends HTTP Alternative Services, in combination with Secondary Certificate Authentication, to enable clients not to disclose the origin to which they intend to connect.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Confidentiality and authentication during communication are primary goals of using TLS to secure traffic on the Internet. However, due to the nature of TLS, certain information is inherently not confidential - notably, the hostname and the corresponding certificate of the origin to which the client is connecting are transferred unencrypted in the Server Name Indication extension [SNI] and the server’s Certificate message [TLS12].

While the client identity can be obscured by using TLS renegotiation immediately after the handshake (in TLS 1.2) or by using TLS 1.3 [TLS13], the server is not afforded such privacy considerations.

Servers may also have wildcard certificates which do not enumerate specific subdomains, but clients will disclose the first subdomain used on a connection via the SNI extension when establishing the connection.

[SNIEncryption] discusses a potential solution to these issues in Section 3, HTTP Co-Tenancy Fronting, but notes both discoverability and server authentication issues with that approach. This document provides a mechanism to address both limitations.

1.1. Usage

In [AltSvc], once a client has received a validated Alternative Service record for an origin, it "SHOULD use that alternative service for all requests to the associated origin as soon as it is available, provided the alternative service information is fresh (Section 2.2) and the security properties of the alternative service protocol are desirable, as compared to the existing connection." However, the client "MUST have reasonable assurances that the alternative service is under control of and valid for the whole origin ... established through use of a TLS-based protocol with the certificate checks defined in [RFC2818]." This causes the origin to be disclosed in the SNI extension while connecting to the alternative, and the origin's certificate to be returned by the alternative, creating the same privacy issues as connecting directly to the origin.

The extension described in Section 2 enables an origin to declare that reasonable assurances should be obtained, not by requesting the desired hostname in the TLS handshake, but by requesting it via [SecondaryCerts]. The validation checks from [RFC2818] are applied to this certificate.

Because the entire exchange happens inside TLS, a passive observer cannot identify the hostname(s) the client might be requesting.

1.2. 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.

The key words "MUST (BUT WE KNOW YOU WON'T)", "SHOULD CONSIDER", "REALLY SHOULD NOT", "OUGHT TO", "WOULD PROBABLY", "MAY WISH TO", "COULD", "POSSIBLE", and "MIGHT" in this document are to be interpreted as described in [RFC6919].

Field definitions are given in Augmented Backus-Naur Form (ABNF), as defined in [RFC5234].

2. The "sni" Alt-Svc Extension

When an origin wishes to nominate a "fronting server", it includes the "sni" parameter in its alternative service entry.

Syntax:

```
sni = ( reg-name / empty-string )
empty-string = DQUOTE DQUOTE
```

"reg-name" is defined in Section 3.2.2 of [RFC3986].

When processing such an alternative, clients SHOULD present the hostname given in the "sni" parameter in the SNI extension during the TLS handshake. If the hostname given is an empty string, clients SHOULD omit the SNI extension from the TLS handshake. The server MUST return a valid certificate which covers at least one of the following:

- The hostname indicated in the SNI extension

- The hostname of the origin that published the alternative

- The hostname used for connecting to the alternative

The client MUST validate the certificate in the handshake for authenticity according to [RFC2818] and ensure that it is valid for at least one of these names. Clients SHOULD NOT accept certificates issued to the IP address of the alternative unless the alternative is specified as an IP literal.

If the certificate is not valid for the origin's hostname, the client MUST NOT make requests to any origin corresponding to this certificate. In this case, the client SHOULD send a "CERTIFICATE_REQUEST" frame including an SNI extension indicating the origin which published the alternative service immediately upon connecting. If no corresponding "CERTIFICATE" frame is presented by the server after a reasonable timeout, or if the server's SETTINGS frame does not include the "SETTINGS_HTTP_CERT_AUTH" setting, the client MUST consider the alternative connection to have failed.

3. Examples

3.1. SNI of Colocated Domain

Suppose a client has received the following Alt-Svc entry for sensitive.example.com in the past:

```
h2="innocence.org:443";ma=2635200;persist=true;sni=innocence.org
```

If the client now wishes to make a request to https://sensitive.example.com/private, it would perform a DNS resolution for innocence.org. The client would then open a TCP connection to the resulting IP address and begin a TLS handshake.

In the client’s TLS handshake, it would request a certificate for the hostname "innocence.org". The TLS server would present such a certificate, issued by an authority trusted by the client. The client will validate the certificate for the name "sensitive.example.com". When validation fails, the client will try to validate the certificate for the name "innocence.org", which will succeed. After validation succeeds, the client will send a "CERTIFICATE_REQUEST" frame asking that the server also authenticate with a certificate for sensitive.example.com.

After receiving the "CERTIFICATE" frame proving possession of a certificate for sensitive.example.com, the client will verify that this certificate is trusted. If so, the client will proceed to send HTTP/2 requests to the server requesting the resource https://sensitive.example.com/private.

3.2. Wildcard Subdomains

Suppose a client has received the following Alt-Svc entry for sensitive.example.com in the past:

```
h2="www.example.com:443";ma=2635200;persist=true;sni=www.example.com
```

If the client now wishes to make a request to https://sensitive.example.com/private, it would perform a DNS resolution for www.example.com, the specified alternative. The client would then open a TCP connection to the resulting IP address and begin a TLS handshake.

In the client’s TLS handshake, it would request a certificate for the hostname www.example.com. The TLS server would present a certificate which included www.example.com as one of the covered hostnames.

Suppose that the certificate with which the server authenticated also contained a Subject Alternative Name of "*.example.com". Because the certificate covers the desired origin, the client would perform validity checks on this certificate.

If the certificate is trusted, the client will proceed to send HTTP/2 requests to the server requesting the resource https://sensitive.example.com/private.

3.3. Omitting SNI

Suppose a client has received the following Alt-Svc entry for sensitive.example.com in the past:

```
h2="alternative.example.com:443";ma=2635200;persist=true;sni=""
```

If the client now wishes to make a request to https://sensitive.example.com/private, it would perform a DNS resolution for alternative.example.com, the specified alternative. The client would then open a TCP connection to the resulting IP address and begin a TLS handshake.

In the client's TLS handshake, it would omit the Server Name Indication extension. The TLS server would present a certificate according to its configured defaults.

The server would supply a certificate that covers sensitive.example.com, for example because it contains a Subject Alternative Name of "*.example.com", and the client would perform validity checks on this certificate.

If the supplied certificate does not cover sensitive.example.com, or is not valid, the client will terminate the connection.

3.4. SNI of Unrelated Domain

Suppose a client has received the following Alt-Svc entry for sensitive.example.com in the past:

```
h2=":443";ma=2635200;persist=true;sni=other.example
```

If the client now wishes to make a request to https://sensitive.example.com/private, it would perform a DNS resolution for sensitive.example.com (the Alt-Svc entry does not specify a different hostname). The client would then open a TCP connection to the resulting IP address and begin a TLS handshake.

In the client's TLS handshake, it would request a certificate for the hostname "other.example". The TLS server does not a have a certificate for this hostname, but it would return a certificate for sensitive.example.com, issued by an authority trusted by the client, and the client will successfully validate the certificate for the name "sensitive.example.com".

Note that an active attacker could identify this server by sending a Client Hello with the same SNI value and observing the certificate the server uses to authenticate. The server could mitigate this by authenticating with a certificate for other.example.

4. Security Considerations

[AltSvc] permits clients to ignore unrecognized parameters. As a result, servers publishing records with the "sni" parameter cannot be assured that clients will not include their origin in the SNI header when connecting to the nominated alternative. If, for security reasons, an origin wishes its identity never to be disclosed when the alternative is being used, an alternative mechanism would be required to ascertain client support before generating the Alt-Svc record.

Clients will need to connect directly to the origin at least once in order to receive the Alt-Svc entry via an HTTP header or "ALTSVC" frame, thus disclosing their use of the origin to the network on the first connection. This could be mitigated by future work defining a way to publish alternative services in a mechanism which can be retrieved confidentially, such as via DNS in combination with [RFC7858] or [DoH].

However, servers which publish Alt-Svc records over unencrypted channels (HTTP connections without TLS) or channels without client authorization (DNS, or publicly accessible HTTP resources) enable active observers to build a map of fronting servers by collecting Alt-Svc advertisements. Servers SHOULD CONSIDER this trade-off in deciding when and how to make Alt-Svc records available to unauthenticated parties.

While concealing information from passive observers is beneficial, low-effort active attacks still exist. If an attacker can collect the actual server identity by sending a Client Hello with the same SNI value, the usefulness of this technique is limited. Server deployments SHOULD reserve sensitive domains for use with Secondary Certificates or conceal them inside wildcards in order to mitigate this.

5. IANA Considerations

The "Hypertext Transfer Protocol (HTTP) Alt-Svc Parameter Registry" defines the name space for parameters, as described in [AltSvc]. It is maintained at http://www.iana.org/assignments/http-alt-svc-parameters [1].

This document registers the following parameter:

Name: "sni"

Specification: This document

6. References

6.1. Normative References

[AltSvc] 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>.

[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>.

[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>.

[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>.

[RFC6919] Barnes, R., Kent, S., and E. Rescorla, "Further Key Words for Use in RFCs to Indicate Requirement Levels", RFC 6919, DOI 10.17487/RFC6919, April 2013, <https://www.rfc-editor.org/info/rfc6919>.

[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>.

[SecondaryCerts]
Bishop, M., Sullivan, N., and M. Thomson, "Secondary Certificate Authentication in HTTP/2", draft-bishop-httpbis-http2-additional-certs-05 (work in progress), October 2017.

[SNI] 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>.

[TLS12] 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>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-28 (work in progress), March 2018.

6.2. Informative References

[DoH] Hoffman, P. and P. McManus, "DNS Queries over HTTPS (DOH)", draft-ietf-doh-dns-over-https-08 (work in progress), May 2018.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <https://www.rfc-editor.org/info/rfc7858>.

[SNIEncryption] Huitema, C. and E. Rescorla, "Issues and Requirements for SNI Encryption in TLS", draft-ietf-tls-sni-encryption-03 (work in progress), May 2018.

6.3. URIs

[1] http://www.iana.org/assignments/http-alt-svc-parameters

Appendix A. Acknowledgements

Conversations with Benjamin Schwartz helped to flesh out this idea.

Author’s Address

Mike Bishop
Akamai

Email: mbishop@evequefou.be

HTTP Working Group K. Oku
Internet-Draft Fastly
Intended status: Experimental Y. Weiss
Expires: January 3, 2019 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

Copyright Notice

Table of Contents

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:

- *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"*"allocated"*"b"/8 rounded up to the nearest integer

5. Add 5 to "bytes"

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

7. Set the first byte to "P"

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

9. Return the "digest-value".

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

2.1.2. Adding a URL to the Digest-Value

Given the following inputs:

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

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

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

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

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

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

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

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

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

9. While "maxcount" is larger than zero:

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

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

    3. While "position_start" < "position_end":

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

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

        3. Add "f" to "position_start".

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

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

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

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

    8. Set "bits" to "dest_fingerprint".

    9. Set "dest_fingerprint" to "fingerprint".

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

    11. Subtract 1 from "maxcount".

10. Subtract "f" from "position_start".

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

12. Let "h1" be "h"

13. Subtract 1 from "maxcount".

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

15. Go to step 7.

2.1.3. Removing a URL to the Digest-Value

Given the following inputs:

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

- "digest-value"

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

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

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

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

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

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

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

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

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

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

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

3. While "position_start" < "position_end":

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

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

    3. Add "f" to "position_start".

2.1.4. Computing a fingerprint value

Given the following inputs:

- "key", an array of characters

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

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

2. Let "h" be the number of bits in "hash-value"

3. Let "fingerprint-value" be 0

4. While "fingerprint-value" is 0 and "h" > "f":

    1. Let "fingerprint-value" be the "f" least significant bits of "hash-value".

    2. Let "hash-value" be the "h"-"f" most significant bits of "hash-value".

    3. Subtract "f" from "h".

5. If "fingerprint-value" is 0, let "fingerprint-value" be 1.

6. Return "fingerprint-value".

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

2.1.5. Computing the key

Given the following inputs:

- "URL", an array of characters

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

2.1.6. Computing a Hash Value

Given the following inputs:

- "key", an array of characters.
- "N", an integer

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

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

2.1.7. Computing an Alternative Hash Value

Given the following inputs:

- "hash1", an integer indicating the previous hash.
- "fingerprint", an integer indicating the fingerprint value.
- "N", an integer indicating the number of entries in the digest.

1. Let "fingerprint-string" be the value of "fingerprint" in base 10, expressed as a string.
2. Let "hash2" be the return value of Section 2.1.6 with "fingerprint-string" and "N" as inputs, XORed with "hash1".
3. Return "hash2".

2.2. Server Behavior

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

- 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 it's first flight data. When such transport (e.g., TLS 1.3 [I-D.ietf-tls-tls13] in full-handshake mode) is used, a client can postpone sending the CACHE_DIGEST frame until it receives a SETTINGS_ACCEPT_CACHE_DIGEST settings value.

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

5. IANA Considerations

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

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 respresents when a request is transmitted through a proxy. The header may represent the cache state of a client or that of a proxy, depending on how the proxy handles the header.

Appendix B. Changes

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

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   I. Grigorik
Internet-Draft   Y. Weiss
Intended status: Experimental   Google
Expires: January 4, 2021   July 3, 2020

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

Abstract

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

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

Note to Readers

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

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

Status of This Memo

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

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

However, traditional proactive content negotiation techniques often mean that user agents send these request headers prolifically. This causes performance concerns (because it creates "bloat" in requests), as well as privacy issues; passively providing such information allows servers to silently fingerprint the user.

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

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

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

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

1.1. Notational Conventions

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

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

2. Client Hint Request Header Fields

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

2.1. Sending Client Hints

User agents choose what Client Hints to send in a request based on their default settings, user configuration, and server preferences expressed in "Accept-CH". The user agent and server can use an opt-in mechanism outlined below to negotiate which header fields need to be sent to allow for efficient content adaption, and optionally use additional mechanisms (e.g., as outlined in [CLIENT-HINTS-INFRASTRUCTURE]) to negotiate delegation policies that control access of third parties to those same header fields. User agents SHOULD require an opt-in to send any hints that are not listed in the low-entropy hint table at [CLIENT-HINTS-INFRASTRUCTURE].

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

2.2. Server Processing of Client Hints

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

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

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

3. Advertising Server Support

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

3.1. The Accept-CH Response Header Field

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

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

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

For example:

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

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

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

3.2. Interaction with Caches

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

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

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

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

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

4. Security Considerations

4.1. Information Exposure

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

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

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

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

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

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

- 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.

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

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

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

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

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

4.2. Deployment and Security Risks

Deployment of new request headers requires several considerations:

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

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

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

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

4.3. Abuse Detection

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

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

5. Cost of Sending Hints

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

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

6. IANA Considerations

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

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

6.1. Accept-CH

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

7. References

7.1. Normative References

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

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

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

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

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

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

7.2. Informative References

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

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

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

7.3. URIs

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

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

[3] https://github.com/httpwg/http-extensions/labels/client-hints

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

Appendix A. Changes

A.1. Since -00

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

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

A.2. Since -01

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

A.3. Since -02

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

A.4. Since -03

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

A.5. Since -04

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

A.6. Since -05

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

A.7. Since -06

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

A.8. Since -07

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

A.9. Since -08

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

A.10. Since -09

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

A.11. Since -10

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

A.12. Since -11

- PR 1134: Re-insert back section.

A.13. Since -12

- PR 1160: AD review.

A.14. Since -13

- PR 1171: Genart review.

A.15. Since -14

- PR 1220: AD review.

Acknowledgements

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

Authors' Addresses

Ilya Grigorik
Google

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

Yoav Weiss
Google

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

HTTP E. Stark
Internet-Draft Google
Intended status: Experimental December 9, 2018
Expires: June 12, 2019

Expect-CT Extension for HTTP
draft-ietf-httpbis-expect-ct-08

Abstract

This document defines a new HTTP header field named Expect-CT, which allows web host operators to instruct user agents to expect valid Signed Certificate Timestamps (SCTs) to be served on connections to these hosts. Expect-CT allows web host operators to discover misconfigurations in their Certificate Transparency deployments. Further, web host operaters can use Expect-CT to ensure that, if a UA which supports Expect-CT accepts a misissued certificate, that certificate will be discoverable in Certificate Transparency logs.

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/expect-ct [3].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines a new HTTP header field that enables UAs to identify web hosts that expect the presence of Signed Certificate Timestamps (SCTs) [I-D.ietf-trans-rfc6962-bis] in subsequent Transport Layer Security (TLS) [RFC8446] connections.

Web hosts that serve the Expect-CT HTTP header field 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 precautions, by using the reporting feature and gradually increasing the time interval during which the UA regards 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.

Expect-CT is similar to HSTS [RFC6797] and HPKP [RFC7469]. HSTS allows web sites to declare themselves accessible only via secure connections, and HPKP allows web sites to declare their cryptographic identifies. Similarly, Expect-CT allows web sites to declare themselves accessible only via connections that are compliant with CT policy.

This Expect-CT specification is compatible with [RFC6962] and [I-D.ietf-trans-rfc6962-bis], but not with future versions of Certificate Transparency. Expect-CT header fields will be ignore from web hosts which use future versions of Certificate Transparency, unless a future version of this document specifies how they should be processed.

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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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 associated with 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" is an abbreviation for "Certificate Transparency Qualified".

- "CT Policy" is an abbreviation for "Certificate Transparency Policy".

- "Effective Expect-CT Date" is the time at which a UA observed a valid Expect-CT header field for a given host.

- "Expect-CT Host" is a conformant host implementing the HTTP server aspects of 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. The term "host" is equivalent to "server" in this specification.

- "Known Expect-CT Host" is an Expect-CT Host that the UA has noted as such. See Section 2.3.2.1 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 [RFC7230].

- "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" response header field is a new field defined in this specification. It is used by a server to indicate that UAs should evaluate connections to the host emitting the header field for CT compliance (Section 2.4).

Figure 1 describes the syntax (Augmented Backus-Naur Form) of the header field, using the grammar defined in [RFC5234] and the rules defined in Section 3.2 of [RFC7230]. The "#" ABNF extension is specified in Section 7 of [RFC7230].

```
Expect-CT           = 1#expect-ct-directive
expect-ct-directive = directive-name [ "=" directive-value ]
directive-name      = token
directive-value     = token / quoted-string
```

Figure 1: Syntax of the Expect-CT header field

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), and Expect-CT is not disabled for local policy reasons (as discussed in Section 2.4.1), the UA MUST process the directives it recognizes.

2.1.1. The report-uri Directive

The OPTIONAL "report-uri" directive indicates the URI to which the UA SHOULD report Expect-CT failures (Section 2.4). 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 [RFC3986].

UAs MUST ignore "report-uri"s that do not use the HTTPS scheme. UAs MUST check Expect-CT compliance when the host in the "report-uri" is a Known Expect-CT Host; similarly, UAs MUST apply HSTS [RFC6797] if the host in the "report-uri" is a Known HSTS Host.

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 to 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.

Note that the report-uri need not necessarily be in the same Internet domain or web origin as the host being reported about. Hosts are in fact encouraged to use a separate host as the report-uri, so that CT failures on the Expect-CT host do not prevent reports from being sent.

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 in the same web browsing session.

2.1.2. 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.

2.1.3. 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.

If a response contains an "Expect-CT" header field, then the response MUST contain an "Expect-CT" header field with a "max-age" directive. (A "max-age" directive need not appear in every "Expect-CT" header field in the response.) 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 [RFC7234].

2.1.4. Examples

The following three examples demonstrate valid Expect-CT response header fields (where the second splits the directives into two field instances):

```
Expect-CT: max-age=86400, enforce

Expect-CT: max-age=86400,enforce
Expect-CT: report-uri="https://foo.example/report"

Expect-CT: max-age=86400,report-uri="https://foo.example/report"
```

Figure 4: Examples of valid Expect-CT response header fields

2.2. Host 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

An Expect-CT Host includes an Expect-CT header field in its response. 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) a 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.

2.3. User Agent Processing Model

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

The UA stores Known Expect-CT Hosts and their associated Expect-CT directives. This data is collectively known as a host's "Expect-CT" metadata".

2.3.1. Missing or Malformed Expect-CT Header Fields

If an HTTP response does not include an Expect-CT header field that conforms to the grammar specified in Section 2.1, then the UA MUST NOT update any Expect-CT metadata.

2.3.2. Expect-CT Header Field Processing

If the UA receives an HTTP response over a secure transport 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 field was received for compliance with the UA's CT Policy, and then process the Expect-CT header field as follows. UAs MUST ignore any Expect-CT header field received in an HTTP response conveyed over non-secure transport.

If the connection does not comply with the UA's CT Policy (i.e., the connection is not CT-qualified), then the UA MUST NOT update any Expect-CT metadata. If the header field includes a "report-uri" directive, the UA SHOULD send a report to the specified "report-uri" (Section 2.3.3).

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

- Note the host as a Known Expect-CT Host if it is not already so noted (see Section 2.3.2.1), or
- 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 a UA receives an Expect-CT header field over a CT-compliant connection which uses a version of Certificate Transparency other than [RFC6962] or [I-D.ietf-trans-rfc6962-bis], the UA MUST ignore the Expect-CT header field and clear any Expect-CT metadata associated with the host.

2.3.2.1. Noting Expect-CT

Upon receipt of the Expect-CT response header field over an error-free TLS connection (with X.509 certificate chain validation as described in [RFC5280], as well as the validation described in Section 2.4), the UA MUST note the host as a Known Expect-CT Host, storing the host's domain name and its associated Expect-CT directives in non-volatile storage.

To note a host as a Known Expect-CT Host, the UA MUST set its Expect-CT metadata in its Known Expect-CT Host cache (as specified in Section 2.3.2.2, using the metadata given in the most recently received valid Expect-CT header field.

For forward compatibility, the UA MUST ignore any unrecognized Expect-CT header field 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.

2.3.2.2. Storage Model

If the substring matching the host production from the Request-URI (of the message to which the host responded) does not exactly match an existing Known Expect-CT Host's domain name, per the matching procedure for a Congruent Match 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, which is the Effective Expect-CT Date plus the value of the "max-age" directive. Alternatively, the UA MAY cache enough information to calculate the Effective Expiration Date. The Effective Expiration Date is calculated from when the UA observed the Expect-CT header field and is independent of when the response was generated.
- 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 field, 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 an Expect-CT Host sets a max- age directive of 90 days in its Expect-CT header field, 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. Reporting

If the UA receives, over a secure transport, an HTTP response that includes an Expect-CT header field with 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.

2.4. Evaluating Expect-CT Connections for CT Compliance

When a UA sets up a TLS connection, the UA determines whether the host is a Known Expect-CT Host according to its Known Expect-CT Host cache. 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 and not treat such hosts as Known Expect-CT hosts.

When a UA connects to a Known Expect-CT Host using a TLS connection, if the TLS 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. However, the check can be skipped for local policy reasons (as discussed in Section 2.4.1), or in the event that other checks cause the UA to terminate the connection before CT compliance is evaluated. For example, a Public Key Pinning failure [RFC7469] could cause the UA to terminate the connection before CT compliance is checked. Similarly, if the UA terminates the connection due to an Expect-CT failure, this could cause the UA to skip subsequent correctness checks. When the CT compliance check is skipped or bypassed, Expect-CT reports (Section 3) will not be sent.

When CT compliance is evaluated for a Known Expect-CT Host, the UA MUST evaluate compliance when setting up the TLS session, before beginning an HTTP conversation over the TLS channel.

If a connection to a Known Expect-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 an error. The UA MAY allow the user to bypass the error, unless connection errors should have no user recourse due to other policies in effect (such as HSTS, as described in Section 12.1 of [RFC6797].

If a connection to a Known Expect-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).

2.4.1. Skipping CT compliance checks

It is acceptable for a UA to skip CT compliance checks 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 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 [RFC8259] object with the following keys and values:

- "date-time": the value for this key indicates the UTC time that the UA observed the CT compliance failure. The value is a string formatted according to Section 5.6, "Internet Date/Time Format", of [RFC3339].

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

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

o "scheme": the value is the scheme with which the UA made the original request that failed the CT compliance check. The value is provided as a string. This key is optional and is assumed to be "https" if not present.

o "effective-expiration-date": the value indicates the Effective Expiration Date (see Section 2.3.2.2) for the Expect-CT Host that failed the CT compliance check, in UTC. The value is provided as a string formatted according to Section 5.6 of [RFC3339] ("Internet Date/Time Format").

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

o "validated-certificate-chain": the value is the certificate chain as constructed by the UA during certificate chain verification. (This may differ from the value of the "served-certificate-chain" key.) The value is provided as an array of strings, which MUST appear in the order matching the chain that the UA validated; each string in the array is the Privacy-Enhanced Mail (PEM) representation of each X.509 certificate as described in [RFC7468]. The first certificate in the chain represents the end-entity certificate being verified. UAs that build certificate chains in more than one way during the validation process SHOULD send the last chain built.

o "scts": the value represents the SCTs (if any) that the UA received for the Expect-CT host and their validation statuses. The value is provided as an array of JSON objects. The SCTs may appear in any order. Each JSON object in the array has the following keys:

  * A "version" key, with an integer value. The UA MUST set this value to "1" if the SCT is in the format defined in Section 3.2 of [RFC6962] and "2" if it is in the format defined in Section 4.5 of [I-D.ietf-trans-rfc6962-bis].

  * The "status" key, with a string value 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 Section 8.2.3 of [I-D.ietf-trans-rfc6962-bis]), or

"invalid" (indicating that the SCT validation failed because of a bad signature or an invalid timestamp).

* The "source" key, with a string value that indicates from where the UA obtained the SCT, as defined in Section 3 of [RFC6962] and Section 6 of [I-D.ietf-trans-rfc6962-bis]. The UA MUST set the value to one of "tls-extension", "ocsp", or "embedded". These correspond to the three methods of delivering SCTs in the TLS handshake that are described in Section 3.3 of [RFC6962].

* The "serialized_sct" key, with a string value. If the value of the "version" key is "1", the UA MUST set this value to the base64 encoded [RFC4648] serialized "SignedCertificateTimestamp" structure from Section 3.2 of [RFC6962]. The base64 encoding is defined in Section 4 of [RFC4648]. If the value of the "version" key is "2", the UA MUST set this value to the base64 encoded [RFC4648] serialized "TransItem" structure representing the SCT, as defined in Section 4.5 of [I-D.ietf-trans-rfc6962-bis].

- "failure-mode": the value indicates whether the Expect-CT report was triggered by an Expect-CT policy in enforce or report-only mode. The value is provided as a string. The UA MUST set this value to "enforce" if the Expect-CT metadata indicates an "enforce" configuration, and "report-only" otherwise.

- "test-report": the value is set to true if the report is being sent by a testing client to verify that the report server behaves correctly. The value is provided as a boolean, and MUST be set to true if the report serves to test the server's behavior and can be discarded.

3.2. Sending a violation report

The UA SHOULD report Expect-CT failures for Known Expect-CT Hosts: that is, when a connection to a Known Expect-CT Host does not comply with the UA's CT Policy and the host's Expect-CT metadata contains a "report-uri".

Additionally, the UA SHOULD report Expect-CT failures for hosts for which it does not have any stored Expect-CT metadata. That is, when the UA connects to a host and receives an Expect-CT header field which contains the "report-uri" directive, the UA SHOULD report an Expect-CT failure if the the connection does not comply with the UA's CT Policy.

The steps to report an Expect-CT failure are 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 Section 3.1.

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. Send an HTTP POST request to "report-uri" with a "Content-Type" header field of "application/expect-ct-report+json", and an entity body consisting of "report body".

The UA MAY perform other operations as part of sending the HTTP POST request, for example sending a CORS preflight as part of [FETCH].

Future versions of this specification may need to modify or extend the Expect-CT report format. They may do so by defining a new top-level key to contain the report, replacing the "expect-ct-report" key. Section 3.3 defines how report servers should handle report formats that they do not support.

3.3. Receiving a violation report

Upon receiving an Expect-CT violation report, the report server MUST respond with a 2xx (Successful) status code if it can parse the request body as valid JSON, the report conforms to the format described in Section 3.1, and it recognizes the scheme, hostname, and port in the "scheme", "hostname", and "port" fields of the report. If the report body cannot be parsed, or the report does not conform to the format described in Section 3.1, or the report server does not expect to receive reports for the scheme, hostname, or port in the report, then the report server MUST respond with a 400 Bad Request status code.

As described in Section 3.2, future versions of this specification may define new report formats that are sent with a different top-level key. If the report server does not recognize the report format, the report server MUST respond with a 501 Not Implemented status code.

If the report's "test-report" key is set to true, the server MAY discard the report without further processing but MUST still return a 2xx (Successful) status code. If the "test-report" key is absent or set to false, the server SHOULD store the report for processing and analysis by the owner of the Expect-CT Host.

4. Usability Considerations

When the UA detects a Known Expect-CT Host in violation of the UA's CT Policy, end users will experience denials of service. It is advisable for UAs to explain to users why they cannot access the Expect-CT Host, e.g., in a user interface that explains that the host's certificate cannot be validated.

5. Authoring Considerations

Expect-CT could be specified as a TLS extension or X.509 certificate extension instead of an HTTP response header field. Using an HTTP header field as the mechanism for Expect-CT introduces a layering mismatch: for example, the software that terminates TLS and validates Certificate Transparency information might know nothing about HTTP. Nevertheless, an HTTP header field was chosen primarily for ease of deployment. In practice, deploying new certificate extensions requires certificate authorities to support them, and new TLS extensions require server software updates, including possibly to servers outside of the site owner's direct control (such as in the case of a third-party CDN). Ease of deployment is a high priority for Expect-CT because it is intended as a temporary transition mechanism for user agents that are transitioning to universal Certificate Transparency requirements.

6. Privacy Considerations

Expect-CT can be used to infer what Certificate Transparency policy a UA is using, by attempting to retrieve specially-configured websites which pass one user agents' policies but not another's. Note that this consideration is true of UAs which enforce CT policies without Expect-CT as well.

Additionally, reports submitted to the "report-uri" could reveal information to a third party about which webpage is being accessed and by which IP address, by using individual "report-uri" values for individually-tracked pages. This information could be leaked even if client-side scripting were disabled.

Implementations store state about Known Expect-CT Hosts, and hence which domains the UA has contacted. Implementations may choose to not store this state subject to local policy (e.g., in the private browsing mode of a web browser).

Violation reports, as noted in Section 3, contain information about the certificate chain that has violated the CT policy. In some cases, such as organization-wide compromise of the end-to-end security of TLS, this may include information about the interception tools and design used by the organization that the organization would otherwise prefer not be disclosed.

Because Expect-CT causes remotely-detectable behavior, it's advisable that UAs offer a way for privacy-sensitive end users to clear currently noted Expect-CT hosts, and allow users to query the current state of Known Expect-CT Hosts.

7. Security Considerations

7.1. Hostile header attacks

When UAs support the Expect-CT header field, it becomes a potential vector for hostile header attacks against site owners. If a site owner uses a certificate issued by a certificate authority which does not embed SCTs nor serve SCTs via OCSP or TLS extension, a malicious server operator or attacker could temporarily reconfigure the host to comply with the UA's CT policy, and add the Expect-CT header field in enforcing mode with a long "max-age". Implementing user agents would note this as an Expect-CT Host (see Section 2.3.2.1). After having done this, the configuration could then be reverted to not comply with the CT policy, prompting failures. Note this scenario would require the attacker to have substantial control over the infrastructure in question, being able to obtain different certificates, change server software, or act as a man-in-the-middle in connections.

Site operators can mitigate this situation by one of: reconfiguring their web server to transmit SCTs using the TLS extension defined in Section 6.5 of [I-D.ietf-trans-rfc6962-bis], obtaining a certificate from an alternative certificate authority which provides SCTs by one of the other methods, or by waiting for the user agents' persisted notation of this as an Expect-CT host to reach its "max-age". User agents may choose to implement mechanisms for users to cure this situation, as noted in Section 4.

7.2. Maximum max-age

There is a security trade-off in that low maximum values provide a narrow window of protection for users that visit the Known Expect-CT Host only infrequently, while high maximum values might result in a denial of service to a UA in the event of a hostile header attack, or simply an error on the part of the site-owner.

There is probably no ideal maximum for the "max-age" directive. Since Expect-CT is primarily a policy-expansion and investigation technology rather than an end-user protection, a value on the order of 30 days (2,592,000 seconds) may be considered a balance between these competing security concerns.

7.3. Amplification attacks

Another kind of hostile header attack uses the "report-uri" mechanism on many hosts not currently exposing SCTs as a method to cause a denial-of-service to the host receiving the reports. If some highly-trafficked websites emitted a non-enforcing Expect-CT header field with a "report-uri", implementing UAs' reports could flood the reporting host. It is noted in Section 2.1.1 that UAs should limit the rate at which they emit reports, but an attacker may alter the Expect-CT header's fields to induce UAs to submit different reports to different URIs to still cause the same effect.

8. IANA Considerations

8.1. Header Field Registry

This document registers the "Expect-CT" header field in the "Permanent Message Header Field Names" registry located at https://www.iana.org/assignments/message-headers [4].

Header field name: Expect-CT

Applicable protocol: http

Status: experimental

Author/Change controller: IETF

Specification document(s): This document

Related information: (empty)

8.2. Media Types Registry

The MIME media type for Expect-CT violation reports is "application/expect-ct-report+json" (which uses the suffix established in [RFC6839]).

Type name: application

Subtype name: expect-ct-report+json

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: binary

Security considerations: See Section 7

Interoperability considerations: n/a

Published specification: This document

Applications that use this media type: UAs that implement Certificate Transparency compliance checks and reporting

Additional information:

Deprecated alias names for this type: n/a

Magic number(s): n/a

File extension(s): n/a

Macintosh file type code(s): n/a

Person & email address to contact for further information: Emily Stark (estark@google.com)

Intended usage: COMMON

Restrictions on usage: none

Author: Emily Stark (estark@google.com)

Change controller: IETF

9. References

9.1. Normative References

[I-D.ietf-trans-rfc6962-bis]
Laurie, B., Langley, A., Kasper, E., Messeri, E., and R. Stradling, "Certificate Transparency Version 2.0", draft-ietf-trans-rfc6962-bis-30 (work in progress), November 2018.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://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, <https://www.rfc-editor.org/info/rfc3986>.

[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>.

[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>.

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

[RFC6839] Hansen, T. and A. Melnikov, "Additional Media Type Structured Syntax Suffixes", RFC 6839, DOI 10.17487/RFC6839, January 2013, <https://www.rfc-editor.org/info/rfc6839>.

[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>.

[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>.

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

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

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

9.2. Informative References

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

9.3. URIs

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

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

[3] https://github.com/httpwg/http-extensions/labels/expect-ct

[4] https://www.iana.org/assignments/message-headers

Appendix A. Changes

A.1. Since -07

- Editorial changes

- Specify that the end-entity certificate appears first in the "validated-certificate-chain" field of an Expect-CT report.

- Define how report format can be extended by future versions of this specification.

- Add optional "scheme" key to report format.

- Specify exact status codes for report server errors.

- Limit report-uris to HTTPS only.

- Note that this version of Expect-CT is only compatible with RFC 6962 and 6962-bis, not any future versions of CT.

A.2. Since -06

- Editorial changes

A.3. Since -05

- Remove SHOULD requirement that UAs disallow certificate error overrides for Known Expect-CT Hosts.

- Remove restriction that Expect-CT Hosts cannot be IP addresses.

- Editorial changes

A.4. Since -04

- Editorial changes

A.5. Since -03

- Editorial changes

A.6. Since -02

- Add concept of test reports and specify that servers must respond with 2xx status codes to valid reports.

- Add "failure-mode" key to reports to allow report servers to distinguish report-only from enforced failures.

A.7. Since -01

- Change SCT reporting format to support both RFC 6962 and 6962-bis SCTs.

A.8. Since -00

- Editorial changes

- o Change Content-Type header of reports to 'application/expect-ct-report+json'
- o Update header field syntax to match convention (issue #327)
- o Reference RFC 6962-bis instead of RFC 6962

Author's Address

Emily Stark
Google

Email: estark@google.com

HTTP P. McManus
Internet-Draft Mozilla
Updates: 6455 (if approved) June 18, 2018
Intended status: Standards Track
Expires: December 20, 2018

Bootstrapping WebSockets with HTTP/2
draft-ietf-httpbis-h2-websockets-07

Abstract

This document defines a mechanism for running the WebSocket Protocol (RFC 6455) over a single stream of an HTTP/2 connection.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The Hypertext Transfer Protocol (HTTP) [RFC7230] provides compatible resource-level semantics across different versions but it does not offer compatibility at the connection management level. Other protocols, such as WebSockets, that rely on connection management details of HTTP must be updated for new versions of HTTP.

The WebSocket Protocol [RFC6455] uses the HTTP/1.1 Upgrade mechanism (Section 6.7 of [RFC7230]) to transition a TCP connection from HTTP into a WebSocket connection. A different approach must be taken with HTTP/2 [RFC7540]. HTTP/2 does not allow connection-wide header fields and status codes such as the Upgrade and Connection request header fields or the 101 (Switching Protocols) response code due to its multiplexing nature. These are all required by the [RFC6455] opening handshake.

Being able to bootstrap WebSockets from HTTP/2 allows one TCP connection to be shared by both protocols and extends HTTP/2's more efficient use of the network to WebSockets.

This document extends the HTTP CONNECT method (as specified for HTTP/2 in Section 8.3 of [RFC7540]). The extension allows the substitution of a new protocol name to connect to rather than the external host normally used by CONNECT. The result is a tunnel on a single HTTP/2 stream that can carry data for WebSockets (or any other protocol). The other streams on the connection may carry more extended CONNECT tunnels, traditional HTTP/2 data, or a mixture of both.

This tunneled stream will be multiplexed with other regular streams on the connection and enjoys the normal priority, cancellation, and flow control features of HTTP/2.

Streams that successfully establish a WebSocket connection using a tunneled stream and the modifications to the opening handshake defined in this document then use the traditional WebSocket Protocol, treating the stream as if were the TCP connection in that specification.

2. Terminology

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

3. The SETTINGS_ENABLE_CONNECT_PROTOCOL SETTINGS Parameter

This document adds a new SETTINGS Parameter to those defined by [RFC7540], Section 6.5.2.

The new parameter name is SETTINGS_ENABLE_CONNECT_PROTOCOL. The value of the parameter MUST be 0 or 1.

Upon receipt of SETTINGS_ENABLE_CONNECT_PROTOCOL with a value of 1, a client MAY use the Extended CONNECT definition of this document when creating new streams. Receipt of this parameter by a server does not have any impact.

A sender MUST NOT send a SETTINGS_ENABLE_CONNECT_PROTOCOL parameter with the value of 0 after previously sending a value of 1.

The use of a SETTINGS Parameter to opt-in to an otherwise incompatible protocol change is a use of "Extending HTTP/2" defined by Section 5.5 of [RFC7540]. Specifically, the addition a new pseudo-header field ":protocol" and the change in meaning of the ":authority" pseudo-header field in Section 4 require opt-in negotiation. If a client were to use the provisions of the extended CONNECT method defined in this document without first receiving a SETTINGS_ENABLE_CONNECT_PROTOCOL parameter, a non-supporting peer would detect a malformed request and generate a stream error (Section 8.1.2.6 of [RFC7540]).

4. The Extended CONNECT Method

Usage of the CONNECT method in HTTP/2 is defined by Section 8.3 of [RFC7540]. This extension modifies the method in the following ways:

- o A new pseudo-header field :protocol MAY be included on request HEADERS indicating the desired protocol to be spoken on the tunnel created by CONNECT. The pseudo-header field is single valued and contains a value from the HTTP Upgrade Token Registry located at https://www.iana.org/assignments/http-upgrade-tokens/http-upgrade-tokens.xhtml

- o On requests that contain the :protocol pseudo-header field, the :scheme and :path pseudo-header fields of the target URI (See Section 5) MUST also be included.

- o On requests bearing the :protocol pseudo-header field, the :authority pseudo-header field is interpreted according to Section 8.1.2.3 of [RFC7540] instead of Section 8.3 of [RFC7540]. In particular, the server MUST NOT create a tunnel to the host indicated by the :authority as it would with a CONNECT method request that was not modified by this extension.

Upon receiving a CONNECT request bearing the :protocol pseudo-header field the server establishes a tunnel to another service of the protocol type indicated by the pseudo-header field. This service may or may not be co-located with the server.

5. Using Extended CONNECT To Bootstrap the WebSocket Protocol

The :protocol pseudo-header field MUST be included in the CONNECT request and it MUST have a value of "websocket" to initiate a WebSocket connection on an HTTP/2 stream. Other HTTP request and response header fields, such as those for manipulating cookies, may be included in the HEADERS with the CONNECT method as usual. This request replaces the GET-based request in [RFC6455] and is used to process the WebSockets opening handshake.

The scheme of the target URI (Section 5.1 of [RFC7230]) MUST be "https" for "wss" schemed WebSockets and "http" for "ws" schemed WebSockets. The remainder of the Target URI is the same as the websocket URI. The websocket URI is still used for proxy autoconfiguration. The security requirements for the HTTP/2 connection used by this specification are established by [RFC7540] for https requests and [RFC8164] for http requests.

[RFC6455] requires the use of Connection and Upgrade header fields that are not part of HTTP/2. They MUST NOT be included in the CONNECT request defined here.

[RFC6455] requires the use of a Host header field which is also not part of HTTP/2. The Host information is conveyed as part of the :authority pseudo-header field which is required on every HTTP/2 transaction.

Implementations using this extended CONNECT to bootstrap WebSockets do not do the processing of the [RFC6455] Sec-WebSocket-Key and Sec-WebSocket-Accept header fields as that functionality has been superseded by the :protocol pseudo-header field.

The Origin [RFC6454], Sec-WebSocket-Version, Sec-WebSocket-Protocol, and Sec-WebSocket-Extensions header fields are used in the CONNECT request and response header fields in the same way as defined in [RFC6455]. Note that HTTP/1 header field names were case-insensitive and HTTP/2 requires they be encoded as lower case.

After successfully processing the opening handshake, the peers should proceed with the WebSocket Protocol [RFC6455] using the HTTP/2 stream from the CONNECT transaction as if it were the TCP connection referred to in [RFC6455]. The state of the WebSocket connection at this point is OPEN as defined by [RFC6455], Section 4.1.

The HTTP/2 stream closure is also analogous to the TCP connection closure of [RFC6455]. Orderly TCP level closures are represented as END_STREAM ([RFC7540], Section 6.1) flags and RST exceptions are represented with the RST_STREAM ([RFC7540], Section 6.4) frame with the CANCEL ([RFC7540], Section 7) error code.

5.1. Example

```
[[ From Client ]]                       [[ From Server ]]

                                        SETTINGS
                                        SETTINGS_ENABLE_CONNECT_[..] = 1

HEADERS + END_HEADERS
:method = CONNECT
:protocol = websocket
:scheme = https
:path = /chat
:authority = server.example.com
sec-websocket-protocol = chat, superchat
sec-websocket-extensions = permessage-deflate
sec-websocket-version = 13
origin = http://www.example.com

                                        HEADERS + END_HEADERS
                                        :status = 200
                                        sec-websocket-protocol = chat

DATA
WebSocket Data

                                        DATA + END_STREAM
                                        WebSocket Data

DATA + END_STREAM
WebSocket Data
```

6. Design Considerations

A more native integration with HTTP/2 is certainly possible with larger additions to HTTP/2. This design was selected to minimize the solution complexity while still addressing the primary concern of running HTTP/2 and WebSockets concurrently.

7. About Intermediaries

This document does not change how WebSockets interacts with HTTP forward proxies. If a client wishing to speak WebSockets connects via HTTP/2 to an HTTP proxy it should continue to use a traditional (i.e. not with a :protocol pseudo-header field) CONNECT to tunnel through that proxy to the WebSocket server via HTTP.

The resulting version of HTTP on that tunnel determines whether WebSockets is initiated directly or via a modified CONNECT request described in this document.

8. Security Considerations

[RFC6455] ensures that non-WebSockets clients, especially XMLHttpRequest based clients, cannot make a WebSocket connection. Its primary mechanism for doing that is the use of Sec- prefixed request header fields that cannot be created by XMLHttpRequest-based clients. This specification addresses that concern in two ways:

- o XMLHttpRequest also prohibits use of the CONNECT method in addition to Sec- prefixed request header fields.

- o The use of a pseudo-header field is something that is connection specific and HTTP/2 does not ever allow to be created outside of the protocol stack.

The security considerations of [RFC6455] section 10 continue to apply to the use of the WebSocket Protocol when using this specification with the exception of 10.8. That section is not relevant because it is specific to the boostrapping handshake that is changed in this document.

9. IANA Considerations

This document establishes an entry for the HTTP/2 Settings Registry that was established by Section 11.3 of [RFC7540].

Name: SETTINGS_ENABLE_CONNECT_PROTOCOL

Code: 0x8

Initial Value: 0

Specification: This document

10. 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>.

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

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/info/rfc6455>.

[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>.

[RFC8164] Nottingham, M. and M. Thomson, "Opportunistic Security for HTTP/2", RFC 8164, DOI 10.17487/RFC8164, May 2017, <https://www.rfc-editor.org/info/rfc8164>.

[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>.

Acknowledgments

The 2017 HTTP Workshop had a very productive discussion that helped determine the key problem and acceptable level of solution complexity.

Author’s Address

Patrick McManus
Mozilla

Email: mcmanus@ducksong.com

HTTP   M. Nottingham
Internet-Draft   Fastly
Intended status: Standards Track   P-H. Kamp
Expires: December 5, 2020   The Varnish Cache Project
                                                         June 3, 2020

Structured Field Values for HTTP
draft-ietf-httpbis-header-structure-19

Abstract

This document describes a set of data types and associated algorithms that are intended to make it easier and safer to define and handle HTTP header and trailer fields, known as "Structured Fields", "Structured Headers", or "Structured Trailers". It is intended for use by specifications of new HTTP fields that wish to use a common syntax that is more restrictive than traditional HTTP field values.

Note to Readers

RFC EDITOR: please remove this section before publication

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

Tests for implementations are collected at https://github.com/httpwg/structured-field-tests [4].

Implementations are tracked at https://github.com/httpwg/wiki/wiki/Structured-Headers [5].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Specifying the syntax of new HTTP header (and trailer) fields is an onerous task; even with the guidance in Section 8.3.1 of [RFC7231], there are many decisions - and pitfalls - for a prospective HTTP field author.

Once a field is defined, bespoke parsers and serializers often need to be written, because each field value has slightly different handling of what looks like common syntax.

This document introduces a set of common data structures for use in definitions of new HTTP field values to address these problems. In particular, it defines a generic, abstract model for them, along with a concrete serialization for expressing that model in HTTP [RFC7230] header and trailer fields.

A HTTP field that is defined as a "Structured Header" or "Structured Trailer" (if the field can be either, it is a "Structured Field") uses the types defined in this specification to define its syntax and basic handling rules, thereby simplifying both its definition by specification writers and handling by implementations.

Additionally, future versions of HTTP can define alternative serializations of the abstract model of these structures, allowing fields that use that model to be transmitted more efficiently without being redefined.

Note that it is not a goal of this document to redefine the syntax of existing HTTP fields; the mechanisms described herein are only intended to be used with fields that explicitly opt into them.

Section 2 describes how to specify a Structured Field.

Section 3 defines a number of abstract data types that can be used in Structured Fields.

Those abstract types can be serialized into and parsed from HTTP field values using the algorithms described in Section 4.

1.1. Intentionally Strict Processing

This specification intentionally defines strict parsing and serialization behaviors using step-by-step algorithms; the only error handling defined is to fail the operation altogether.

It is designed to encourage faithful implementation and therefore good interoperability. Therefore, an implementation that tried to be helpful by being more tolerant of input would make interoperability worse, since that would create pressure on other implementations to implement similar (but likely subtly different) workarounds.

In other words, strict processing is an intentional feature of this specification; it allows non-conformant input to be discovered and corrected by the producer early, and avoids both interoperability and security issues that might otherwise result.

Note that as a result of this strictness, if a field is appended to by multiple parties (e.g., intermediaries, or different components in the sender), an error in one party's value is likely to cause the entire field value to fail parsing.

1.2. 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 algorithms to specify parsing and serialization behaviors, and the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] to illustrate expected syntax in HTTP header fields. In doing so, it uses the VCHAR, SP, DIGIT, ALPHA and DQUOTE rules from [RFC5234]. It also includes the tchar and OWS rules from [RFC7230].

When parsing from HTTP fields, implementations MUST have behavior that is indistinguishable from following the algorithms. If there is disagreement between the parsing algorithms and ABNF, the specified algorithms take precedence.

For serialization to HTTP fields, the ABNF illustrates their expected wire representations, and the algorithms define the recommended way to produce them. Implementations MAY vary from the specified behavior so long as the output is still correctly handled by the parsing algorithm.

2. Defining New Structured Fields

To specify a HTTP field as a Structured Field, its authors needs to:

- Normatively reference this specification. Recipients and generators of the field need to know that the requirements of this document are in effect.

- o Identify whether the field is a Structured Header (i.e., it can only be used in the header section - the common case), a Structured Trailer (only in the trailer section), or a Structured Field (both).

- o Specify the type of the field value; either List (Section 3.1), Dictionary (Section 3.2), or Item (Section 3.3).

- o Define the semantics of the field value.

- o Specify any additional constraints upon the field value, as well as the consequences when those constraints are violated.

Typically, this means that a field definition will specify the top-level type - List, Dictionary or Item - and then define its allowable types, and constraints upon them. For example, a header defined as a List might have all Integer members, or a mix of types; a header defined as an Item might allow only Strings, and additionally only strings beginning with the letter "Q", or strings in lowercase. Likewise, Inner Lists (Section 3.1.1) are only valid when a field definition explicitly allows them.

When parsing fails, the entire field is ignored (see Section 4.2); in most situations, violating field-specific constraints should have the same effect. Thus, if a header is defined as an Item and required to be an Integer, but a String is received, the field will by default be ignored. If the field requires different error handling, this should be explicitly specified.

Both Items and Inner Lists allow parameters as an extensibility mechanism; this means that values can later be extended to accommodate more information, if need be. To preserve forward compatibility, field specifications are discouraged from defining the presence of an unrecognized Parameter as an error condition.

To further assure that this extensibility is available in the future, and to encourage consumers to use a complete parser implementation, a field definition can specify that "grease" Parameters be added by senders. A specification could stipulate that all Parameters that fit a defined pattern are reserved for this use and then encourage them to be sent on some portion of requests. This helps to discourage recipients from writing a parser that does not account for Parameters.

Specifications that use Dictionaries can also allow for forward compatibility by requiring that the presence of - as well as value and type associated with - unknown members be ignored. Later specifications can then add additional members, specifying constraints on them as appropriate.

An extension to a structured field can then require that an entire field value be ignored by a recipient that understands the extension if constraints on the value it defines are not met.

A field definition cannot relax the requirements of this specification because doing so would preclude handling by generic software; they can only add additional constraints (for example, on the numeric range of Integers and Decimals, the format of Strings and Tokens, the types allowed in a Dictionary's values, or the number of Items in a List). Likewise, field definitions can only use this specification for the entire field value, not a portion thereof.

This specification defines minimums for the length or number of various structures supported by implementations. It does not specify maximum sizes in most cases, but authors should be aware that HTTP implementations do impose various limits on the size of individual fields, the total number of fields, and/or the size of the entire header or trailer section.

Specifications can refer to a field name as a "structured header name", "structured trailer name" or "structured field name" as appropriate. Likewise, they can refer its field value as a "structured header value", "structured trailer value" or "structured field value" as necessary. Field definitions are encouraged to use the ABNF rules beginning with "sf-" defined in this specification; other rules in this specification are not intended for their use.

For example, a fictitious Foo-Example header field might be specified as:

--8<--
42. Foo-Example Header

The Foo-Example HTTP header field conveys information about how much Foo the message has.

Foo-Example is a Item Structured Header [RFCxxxx]. Its value MUST be an Integer (Section Y.Y of [RFCxxxx]). Its ABNF is:

```
Foo-Example = sf-integer
```

Its value indicates the amount of Foo in the message, and MUST be between 0 and 10, inclusive; other values MUST cause the entire header field to be ignored.

The following parameters are defined:
* A Parameter whose name is "foourl", and whose value is a String (Section Y.Y of [RFCxxxx]), conveying the Foo URL for the message. See below for processing requirements.

"foourl" contains a URI-reference (Section 4.1 of [RFC3986]). If its value is not a valid URI-reference, the entire header field MUST be ignored. If its value is a relative reference (Section 4.2 of [RFC3986]), it MUST be resolved (Section 5 of [RFC3986]) before being used.

For example:

```
Foo-Example: 2; foourl="https://foo.example.com/"
```

-->8--

3. Structured Data Types

This section defines the abstract types for Structured Fields. The ABNF provided represents the on-wire format in HTTP field values.

In summary:

o There are three top-level types that a HTTP field can be defined as: Lists, Dictionaries, and Items.

o Lists and Dictionaries are containers; their members can be Items or Inner Lists (which are themselves arrays of Items).

o Both Items and Inner Lists can be parameterized with key/value pairs.

3.1. Lists

Lists are arrays of zero or more members, each of which can be an Item (Section 3.3) or an Inner List (Section 3.1.1), both of which can be Parameterized (Section 3.1.2).

The ABNF for Lists in HTTP fields is:

```
sf-list       = list-member *( OWS "," OWS list-member )
list-member   = sf-item / inner-list
```

Each member is separated by a comma and optional whitespace. For example, a field whose value is defined as a List of Strings could look like:

```
Example-StrList: "foo", "bar", "It was the best of times."
```

An empty List is denoted by not serializing the field at all. This implies that fields defined as Lists have a default empty value.

Note that Lists can have their members split across multiple lines inside a header or trailer section, as per Section 3.2.2 of [RFC7230]; for example, the following are equivalent:

```
Example-Hdr: foo, bar
```

and

```
Example-Hdr: foo
Example-Hdr: bar
```

However, individual members of a List cannot be safely split between across lines; see Section 4.2 for details.

Parsers MUST support Lists containing at least 1024 members. Field specifications can constrain the types and cardinality of individual List values as they require.

3.1.1. Inner Lists

An Inner List is an array of zero or more Items (Section 3.3). Both the individual Items and the Inner List itself can be Parameterized (Section 3.1.2).

The ABNF for Inner Lists is:

```
inner-list    = "(" *SP [ sf-item *( 1*SP sf-item ) *SP ] ")"
                parameters
```

Inner Lists are denoted by surrounding parenthesis, and have their values delimited by one or more spaces. A field whose value is defined as a List of Inner Lists of Strings could look like:

```
Example-StrListList: ("foo" "bar"), ("baz"), ("bat" "one"), ()
```

Note that the last member in this example is an empty Inner List.

A header field whose value is defined as a List of Inner Lists with Parameters at both levels could look like:

```
Example-ListListParam: ("foo"; a=1;b=2);lvl=5, ("bar" "baz");lvl=1
```

Parsers MUST support Inner Lists containing at least 256 members. Field specifications can constrain the types and cardinality of individual Inner List members as they require.

3.1.2. Parameters

Parameters are an ordered map of key-value pairs that are associated with an Item (Section 3.3) or Inner List (Section 3.1.1). The keys are unique within the scope the Parameters they occur within, and the values are bare items (i.e., they themselves cannot be parameterized; see Section 3.3).

The ABNF for Parameters is:

```
parameters    = *( ";" *SP parameter )
parameter     = param-name [ "=" param-value ]
param-name    = key
key           = ( lcalpha / "*" )
                *( lcalpha / DIGIT / "_" / "-" / "." / "*" )
lcalpha       = %x61-7A ; a-z
param-value   = bare-item
```

Note that Parameters are ordered as serialized, and Parameter keys cannot contain uppercase letters. A parameter is separated from its Item or Inner List and other parameters by a semicolon. For example:

```
Example-ParamList: abc;a=1;b=2; cde_456, (ghi;jk=4 l);q="9";r=w
```

Parameters whose value is Boolean (see Section 3.3.6) true MUST omit that value when serialized. For example, the "a" parameter here is true, while the "b" parameter is false:

```
Example-Int: 1; a; b=?0
```

Note that this requirement is only on serialization; parsers are still required to correctly handle the true value when it appears in a parameter.

Parsers MUST support at least 256 parameters on an Item or Inner List, and support parameter keys with at least 64 characters. Field specifications can constrain the order of individual Parameters, as well as their values' types as required.

3.2. Dictionaries

Dictionaries are ordered maps of name-value pairs, where the names are short textual strings and the values are Items (Section 3.3) or arrays of Items, both of which can be Parameterized (Section 3.1.2). There can be zero or more members, and their names are unique in the scope of the Dictionary they occur within.

Implementations MUST provide access to Dictionaries both by index and by name. Specifications MAY use either means of accessing the members.

The ABNF for Dictionaries is:

```
sf-dictionary  = dict-member *( OWS "," OWS dict-member )
dict-member    = member-name [ "=" member-value ]
member-name    = key
member-value   = sf-item / inner-list
```

Members are ordered as serialized, and separated by a comma with optional whitespace. Member names cannot contain uppercase characters. Names and values are separated by "=" (without whitespace). For example:

```
Example-Dict: en="Applepie", da=:w4ZibGV0w6ZydGU=:
```

Note that in this example, the final "=" is due to the inclusion of a Byte Sequence; see Section 3.3.5.

Members whose value is Boolean (see Section 3.3.6) true MUST omit that value when serialized. For example, here both "b" and "c" are true:

```
Example-Dict: a=?0, b, c; foo=bar
```

Note that this requirement is only on serialization; parsers are still required to correctly handle the true Boolean value when it appears in Dictionary values.

A Dictionary with a member whose value is an Inner List of Tokens:

```
Example-DictList: rating=1.5, feelings=(joy sadness)
```

A Dictionary with a mix of Items and Inner Lists, some with Parameters:

```
Example-MixDict: a=(1 2), b=3, c=4;aa=bb, d=(5 6);valid
```

As with lists, an empty Dictionary is represented by omitting the entire field. This implies that fields defined as Dictionaries have a default empty value.

Typically, a field specification will define the semantics of Dictionaries by specifying the allowed type(s) for individual members by their names, as well as whether their presence is required or optional. Recipients MUST ignore names that are undefined or unknown, unless the field's specification specifically disallows them.

Note that Dictionaries can have their members split across multiple lines inside a header or trailer section; for example, the following are equivalent:

```
Example-Hdr: foo=1, bar=2
```

and

```
Example-Hdr: foo=1
Example-Hdr: bar=2
```

However, individual members of a Dictionary cannot be safely split between lines; see Section 4.2 for details.

Parsers MUST support Dictionaries containing at least 1024 name/value pairs, and names with at least 64 characters. Field specifications can constrain the order of individual Dictionary members, as well as their values' types as required.

3.3. Items

An Item can be a Integer (Section 3.3.1), Decimal (Section 3.3.2), String (Section 3.3.3), Token (Section 3.3.4), Byte Sequence (Section 3.3.5), or Boolean (Section 3.3.6). It can have associated Parameters (Section 3.1.2).

The ABNF for Items is:

```
sf-item   = bare-item parameters
bare-item = sf-integer / sf-decimal / sf-string / sf-token
            / sf-binary / sf-boolean
```

For example, a header field that is defined to be an Item that is an Integer might look like:

```
Example-IntItemHeader: 5
```

or with Parameters:

```
Example-IntItem: 5; foo=bar
```

3.3.1. Integers

Integers have a range of -999,999,999,999,999 to 999,999,999,999,999 inclusive (i.e., up to fifteen digits, signed), for IEEE 754 compatibility ([IEEE754]).

The ABNF for Integers is:

```
sf-integer = ["-"] 1*15DIGIT
```

For example:

```
Example-Integer: 42
```

Integers larger than 15 digits can be supported in a variety of ways; for example, by using a String (Section 3.3.3), Byte Sequence (Section 3.3.5), or a parameter on an Integer that acts as a scaling factor.

While it is possible to serialise Integers with leading zeros (e.g., "0002", "-01") and signed zero ("-0"), these distinctions may not be preserved by implementations.

Note that commas in Integers are used in this section’s prose only for readability; they are not valid in the wire format.

3.3.2. Decimals

Decimals are numbers with an integer and a fractional component. The integer component has at most 12 digits; the fractional component has at most three digits.

The ABNF for decimals is:

```
sf-decimal  = ["-"] 1*12DIGIT "." 1*3DIGIT
```

For example, a header whose value is defined as a Decimal could look like:

```
Example-Decimal: 4.5
```

While it is possible to serialise Decimals with leading zeros (e.g., "0002.5", "-01.334"), trailing zeros (e.g., "5.230", "-0.40"), and signed zero (e.g., "-0.0"), these distinctions may not be preserved by implementations.

Note that the serialisation algorithm (Section 4.1.5) rounds input with more than three digits of precision in the fractional component. If an alternative rounding strategy is desired, this should be specified by the header definition to occur before serialisation.

3.3.3. Strings

Strings are zero or more printable ASCII [RFC0020] characters (i.e., the range %x20 to %x7E). Note that this excludes tabs, newlines, carriage returns, etc.

The ABNF for Strings is:

```
sf-string = DQUOTE *chr DQUOTE
chr       = unescaped / escaped
unescaped = %x20-21 / %x23-5B / %x5D-7E
escaped   = "\" ( DQUOTE / "\" )
```

Strings are delimited with double quotes, using a backslash ("\") to escape double quotes and backslashes. For example:

```
Example-String: "hello world"
```

Note that Strings only use DQUOTE as a delimiter; single quotes do not delimit Strings. Furthermore, only DQUOTE and "\" can be escaped; other characters after "\" MUST cause parsing to fail.

Unicode is not directly supported in Strings, because it causes a number of interoperability issues, and - with few exceptions - field values do not require it.

When it is necessary for a field value to convey non-ASCII content, a Byte Sequence (Section 3.3.5) can be specified, along with a character encoding (preferably [UTF-8]).

Parsers MUST support Strings (after any decoding) with at least 1024 characters.

3.3.4. Tokens

Tokens are short textual words; their abstract model is identical to their expression in the HTTP field value serialization.

The ABNF for Tokens is:

```
sf-token = ( ALPHA / "*" ) *( tchar / ":" / "/" )
```

For example:

```
Example-Token: foo123/456
```

Parsers MUST support Tokens with at least 512 characters.

Note that Token allows the same characters as the "token" ABNF rule defined in [RFC7230], with the exceptions that the first character is required to be either ALPHA or "*", and ":" and "/" are also allowed in subsequent characters.

3.3.5. Byte Sequences

Byte Sequences can be conveyed in Structured Fields.

The ABNF for a Byte Sequence is:

```
sf-binary = ":" *(base64) ":"
base64    = ALPHA / DIGIT / "+" / "/" / "="
```

A Byte Sequence is delimited with colons and encoded using base64 ([RFC4648], Section 4). For example:

```
Example-Binary: :cHJldGVuZCB0aGlzIGlzIGJpbmFyeSBjb250ZW50Lg==:
```

Parsers MUST support Byte Sequences with at least 16384 octets after decoding.

3.3.6. Booleans

Boolean values can be conveyed in Structured Fields.

The ABNF for a Boolean is:

```
sf-boolean = "?" boolean
boolean    = "0" / "1"
```

A Boolean is indicated with a leading "?" character followed by a "1" for a true value or "0" for false. For example:

Example-Bool: ?1

Note that in Dictionary (Section 3.2) and Parameter (Section 3.1.2) values, Boolean true is indicated by omitting the value.

4. Working With Structured Fields in HTTP

This section defines how to serialize and parse Structured Fields in textual HTTP field values and other encodings compatible with them (e.g., in HTTP/2 [RFC7540] before compression with HPACK [RFC7541]).

4.1. Serializing Structured Fields

Given a structure defined in this specification, return an ASCII string suitable for use in a HTTP field value.

1. If the structure is a Dictionary or List and its value is empty (i.e., it has no members), do not serialize the field at all (i.e., omit both the field-name and field-value).

2. If the structure is a List, let output_string be the result of running Serializing a List (Section 4.1.1) with the structure.

3. Else if the structure is a Dictionary, let output_string be the result of running Serializing a Dictionary (Section 4.1.2) with the structure.

4. Else if the structure is an Item, let output_string be the result of running Serializing an Item (Section 4.1.3) with the structure.

5. Else, fail serialization.

6. Return output_string converted into an array of bytes, using ASCII encoding [RFC0020].

4.1.1. Serializing a List

Given an array of (member_value, parameters) tuples as input_list, return an ASCII string suitable for use in a HTTP field value.

1. Let output be an empty string.

2. For each (member_value, parameters) of input_list:

    1. If member_value is an array, append the result of running Serializing an Inner List (Section 4.1.1.1) with (member_value, parameters) to output.

    2. Otherwise, append the result of running Serializing an Item (Section 4.1.3) with (member_value, parameters) to output.

    3. If more member_values remain in input_list:

        1. Append "," to output.

        2. Append a single SP to output.

3. Return output.

4.1.1.1. Serializing an Inner List

Given an array of (member_value, parameters) tuples as inner_list, and parameters as list_parameters, return an ASCII string suitable for use in a HTTP field value.

1. Let output be the string "(".

2. For each (member_value, parameters) of inner_list:

    1. Append the result of running Serializing an Item (Section 4.1.3) with (member_value, parameters) to output.

    2. If more values remain in inner_list, append a single SP to output.

3. Append ")" to output.

4. Append the result of running Serializing Parameters (Section 4.1.1.2) with list_parameters to output.

5. Return output.

4.1.1.2. Serializing Parameters

Given an ordered Dictionary as input_parameters (each member having a param_name and a param_value), return an ASCII string suitable for use in a HTTP field value.

1. Let output be an empty string.

2. For each param_name with a value of param_value in input_parameters:

    1. Append ";" to output.

    2. Append the result of running Serializing a Key (Section 4.1.1.3) with param_name to output.

    3. If param_value is not Boolean true:

        1. Append "=" to output.

        2. Append the result of running Serializing a bare Item (Section 4.1.3.1) with param_value to output.

3. Return output.

4.1.1.3. Serializing a Key

Given a key as input_key, return an ASCII string suitable for use in a HTTP field value.

1. Convert input_key into a sequence of ASCII characters; if conversion fails, fail serialization.

2. If input_key contains characters not in lcalpha, DIGIT, "_", "-", ".", or "*" fail serialization.

3. If the first character of input_key is not lcalpha or "*", fail serialization.

4. Let output be an empty string.

5. Append input_key to output.

6. Return output.

4.1.2. Serializing a Dictionary

Given an ordered Dictionary as input_dictionary (each member having a member_name and a tuple value of (member_value, parameters)), return an ASCII string suitable for use in a HTTP field value.

1. Let output be an empty string.

2. For each member_name with a value of (member_value, parameters) in input_dictionary:

    1. Append the result of running Serializing a Key (Section 4.1.1.3) with member's member_name to output.

    2. If member_value is Boolean true:

1. Append the result of running Serializing Parameters (Section 4.1.1.2) with parameters to output.

3. Otherwise:

1. Append "=" to output.

2. If member_value is an array, append the result of running Serializing an Inner List (Section 4.1.1.1) with (member_value, parameters) to output.

3. Otherwise, append the result of running Serializing an Item (Section 4.1.3) with (member_value, parameters) to output.

4. If more members remain in input_dictionary:

1. Append "," to output.

2. Append a single SP to output.

3. Return output.

4.1.3. Serializing an Item

Given an Item as bare_item and Parameters as item_parameters, return an ASCII string suitable for use in a HTTP field value.

1. Let output be an empty string.

2. Append the result of running Serializing a Bare Item Section 4.1.3.1 with bare_item to output.

3. Append the result of running Serializing Parameters Section 4.1.1.2 with item_parameters to output.

4. Return output.

4.1.3.1. Serializing a Bare Item

Given an Item as input_item, return an ASCII string suitable for use in a HTTP field value.

1. If input_item is an Integer, return the result of running Serializing an Integer (Section 4.1.4) with input_item.

2. If input_item is a Decimal, return the result of running Serializing a Decimal (Section 4.1.5) with input_item.

3. If input_item is a String, return the result of running Serializing a String (Section 4.1.6) with input_item.

4. If input_item is a Token, return the result of running Serializing a Token (Section 4.1.7) with input_item.

5. If input_item is a Boolean, return the result of running Serializing a Boolean (Section 4.1.9) with input_item.

6. If input_item is a Byte Sequence, return the result of running Serializing a Byte Sequence (Section 4.1.8) with input_item.

7. Otherwise, fail serialization.

4.1.4. Serializing an Integer

Given an Integer as input_integer, return an ASCII string suitable for use in a HTTP field value.

1. If input_integer is not an integer in the range of -999,999,999,999,999 to 999,999,999,999,999 inclusive, fail serialization.

2. Let output be an empty string.

3. If input_integer is less than (but not equal to) 0, append "-" to output.

4. Append input_integer's numeric value represented in base 10 using only decimal digits to output.

5. Return output.

4.1.5. Serializing a Decimal

Given a decimal number as input_decimal, return an ASCII string suitable for use in a HTTP field value.

1. If input_decimal is not a decimal number, fail serialization.

2. If input_decimal has more than three significant digits to the right of the decimal point, round it to three decimal places, rounding the final digit to the nearest value, or to the even value if it is equidistant.

3. If input_decimal has more than 12 significant digits to the left of the decimal point after rounding, fail serialization.

4. Let output be an empty string.

5. If input_decimal is less than (but not equal to) 0, append "-" to output.

6. Append input_decimal's integer component represented in base 10 (using only decimal digits) to output; if it is zero, append "0".

7. Append "." to output.

8. If input_decimal's fractional component is zero, append "0" to output.

9. Otherwise, append the significant digits of input_decimal's fractional component represented in base 10 (using only decimal digits) to output.

10. Return output.

4.1.6. Serializing a String

Given a String as input_string, return an ASCII string suitable for use in a HTTP field value.

1. Convert input_string into a sequence of ASCII characters; if conversion fails, fail serialization.

2. If input_string contains characters in the range %x00-1f or %x7f (i.e., not in VCHAR or SP), fail serialization.

3. Let output be the string DQUOTE.

4. For each character char in input_string:

    1. If char is "\" or DQUOTE:

        1. Append "\" to output.

    2. Append char to output.

5. Append DQUOTE to output.

6. Return output.

4.1.7. Serializing a Token

Given a Token as input_token, return an ASCII string suitable for use in a HTTP field value.

1. Convert input_token into a sequence of ASCII characters; if conversion fails, fail serialization.

2. If the first character of input_token is not ALPHA or "*", or the remaining portion contains a character not in tchar, ":" or "/", fail serialization.

3. Let output be an empty string.

4. Append input_token to output.

5. Return output.

4.1.8. Serializing a Byte Sequence

Given a Byte Sequence as input_bytes, return an ASCII string suitable for use in a HTTP field value.

1. If input_bytes is not a sequence of bytes, fail serialization.

2. Let output be an empty string.

3. Append ":" to output.

4. Append the result of base64-encoding input_bytes as per [RFC4648], Section 4, taking account of the requirements below.

5. Append ":" to output.

6. Return output.

The encoded data is required to be padded with "=", as per [RFC4648], Section 3.2.

Likewise, encoded data SHOULD have pad bits set to zero, as per [RFC4648], Section 3.5, unless it is not possible to do so due to implementation constraints.

4.1.9. Serializing a Boolean

Given a Boolean as input_boolean, return an ASCII string suitable for use in a HTTP field value.

1. If input_boolean is not a boolean, fail serialization.

2. Let output be an empty string.

3. Append "?" to output.

4. If input_boolean is true, append "1" to output.

5. If input_boolean is false, append "0" to output.

6. Return output.

4.2. Parsing Structured Fields

When a receiving implementation parses HTTP fields that are known to be Structured Fields, it is important that care be taken, as there are a number of edge cases that can cause interoperability or even security problems. This section specifies the algorithm for doing so.

Given an array of bytes input_bytes that represents the chosen field's field-value (which is empty if that field is not present), and field_type (one of "dictionary", "list", or "item"), return the parsed header value.

1. Convert input_bytes into an ASCII string input_string; if conversion fails, fail parsing.

2. Discard any leading SP characters from input_string.

3. If field_type is "list", let output be the result of running Parsing a List (Section 4.2.1) with input_string.

4. If field_type is "dictionary", let output be the result of running Parsing a Dictionary (Section 4.2.2) with input_string.

5. If field_type is "item", let output be the result of running Parsing an Item (Section 4.2.3) with input_string.

6. Discard any leading SP characters from input_string.

7. If input_string is not empty, fail parsing.

8. Otherwise, return output.

When generating input_bytes, parsers MUST combine all field lines in the same section (header or trailer) that case-insensitively match the field name into one comma-separated field-value, as per

[RFC7230], Section 3.2.2; this assures that the entire field value is processed correctly.

For Lists and Dictionaries, this has the effect of correctly concatenating all of the field's lines, as long as individual members of the top-level data structure are not split across multiple header instances. The parsing algorithms for both types allow tab characters, since these might be used to combine field lines by some implementations.

Strings split across multiple field lines will have unpredictable results, because comma(s) and whitespace inserted upon combination will become part of the string output by the parser. Since concatenation might be done by an upstream intermediary, the results are not under the control of the serializer or the parser, even when they are both under the control of the same party.

Tokens, Integers, Decimals and Byte Sequences cannot be split across multiple field lines because the inserted commas will cause parsing to fail.

Parsers MAY fail when processing a field value spread across multiple field lines, when one of those lines does not parse as that field. For example, a parsing handling an Example-String field that's defined as a sf-string is allowed to fail when processing this field section:

```
Example-String: "foo
Example-String: bar"
```

If parsing fails - including when calling another algorithm - the entire field value MUST be ignored (i.e., treated as if the field were not present in the section). This is intentionally strict, to improve interoperability and safety, and specifications referencing this document are not allowed to loosen this requirement.

Note that this requirement does not apply to an implementation that is not parsing the field; for example, an intermediary is not required to strip a failing field from a message before forwarding it.

4.2.1. Parsing a List

Given an ASCII string as input_string, return an array of (item_or_inner_list, parameters) tuples. input_string is modified to remove the parsed value.

1. Let members be an empty array.

2. While input_string is not empty:

    1. Append the result of running Parsing an Item or Inner List (Section 4.2.1.1) with input_string to members.

    2. Discard any leading OWS characters from input_string.

    3. If input_string is empty, return members.

    4. Consume the first character of input_string; if it is not ",", fail parsing.

    5. Discard any leading OWS characters from input_string.

    6. If input_string is empty, there is a trailing comma; fail parsing.

3. No structured data has been found; return members (which is empty).

4.2.1.1. Parsing an Item or Inner List

Given an ASCII string as input_string, return the tuple (item_or_inner_list, parameters), where item_or_inner_list can be either a single bare item, or an array of (bare_item, parameters) tuples. input_string is modified to remove the parsed value.

1. If the first character of input_string is "(", return the result of running Parsing an Inner List (Section 4.2.1.2) with input_string.

2. Return the result of running Parsing an Item (Section 4.2.3) with input_string.

4.2.1.2. Parsing an Inner List

Given an ASCII string as input_string, return the tuple (inner_list, parameters), where inner_list is an array of (bare_item, parameters) tuples. input_string is modified to remove the parsed value.

1. Consume the first character of input_string; if it is not "(", fail parsing.

2. Let inner_list be an empty array.

3. While input_string is not empty:

    1. Discard any leading SP characters from input_string.

2. If the first character of input_string is ")":

    1. Consume the first character of input_string.

    2. Let parameters be the result of running Parsing Parameters (Section 4.2.3.2) with input_string.

    3. Return the tuple (inner_list, parameters).

3. Let item be the result of running Parsing an Item (Section 4.2.3) with input_string.

4. Append item to inner_list.

5. If the first character of input_string is not SP or ")", fail parsing.

4. The end of the inner list was not found; fail parsing.

4.2.2. Parsing a Dictionary

Given an ASCII string as input_string, return an ordered map whose values are (item_or_inner_list, parameters) tuples. input_string is modified to remove the parsed value.

1. Let dictionary be an empty, ordered map.

2. While input_string is not empty:

    1. Let this_key be the result of running Parsing a Key (Section 4.2.3.3) with input_string.

    2. If the first character of input_string is "=":

        1. Consume the first character of input_string.

        2. Let member be the result of running Parsing an Item or Inner List (Section 4.2.1.1) with input_string.

    3. Otherwise:

        1. Let value be Boolean true.

        2. Let parameters be the result of running Parsing Parameters Section 4.2.3.2 with input_string.

        3. Let member be the tuple (value, parameters).

    4. Add name this_key with value member to dictionary. If dictionary already contains a name this_key (comparing character-for-character), overwrite its value.

    5. Discard any leading OWS characters from input_string.

    6. If input_string is empty, return dictionary.

    7. Consume the first character of input_string; if it is not ",", fail parsing.

    8. Discard any leading OWS characters from input_string.

    9. If input_string is empty, there is a trailing comma; fail parsing.

3. No structured data has been found; return dictionary (which is empty).

Note that when duplicate Dictionary keys are encountered, this has the effect of ignoring all but the last instance.

4.2.3. Parsing an Item

Given an ASCII string as input_string, return a (bare_item, parameters) tuple. input_string is modified to remove the parsed value.

1. Let bare_item be the result of running Parsing a Bare Item (Section 4.2.3.1) with input_string.

2. Let parameters be the result of running Parsing Parameters (Section 4.2.3.2) with input_string.

3. Return the tuple (bare_item, parameters).

4.2.3.1. Parsing a Bare Item

Given an ASCII string as input_string, return a bare Item. input_string is modified to remove the parsed value.

1. If the first character of input_string is a "-" or a DIGIT, return the result of running Parsing an Integer or Decimal (Section 4.2.4) with input_string.

2. If the first character of input_string is a DQUOTE, return the result of running Parsing a String (Section 4.2.5) with input_string.

3. If the first character of input_string is ":", return the result of running Parsing a Byte Sequence (Section 4.2.7) with input_string.

4. If the first character of input_string is "?", return the result of running Parsing a Boolean (Section 4.2.8) with input_string.

5. If the first character of input_string is an ALPHA or "*", return the result of running Parsing a Token (Section 4.2.6) with input_string.

6. Otherwise, the item type is unrecognized; fail parsing.

4.2.3.2. Parsing Parameters

Given an ASCII string as input_string, return an ordered map whose values are bare Items. input_string is modified to remove the parsed value.

1. Let parameters be an empty, ordered map.

2. While input_string is not empty:

    1. If the first character of input_string is not ";", exit the loop.

    2. Consume a ";" character from the beginning of input_string.

    3. Discard any leading SP characters from input_string.

    4. let param_name be the result of running Parsing a Key (Section 4.2.3.3) with input_string.

    5. Let param_value be Boolean true.

    6. If the first character of input_string is "=":

        1. Consume the "=" character at the beginning of input_string.

        2. Let param_value be the result of running Parsing a Bare Item (Section 4.2.3.1) with input_string.

    7. Append key param_name with value param_value to parameters. If parameters already contains a name param_name (comparing character-for-character), overwrite its value.

3. Return parameters.

Note that when duplicate Parameter keys are encountered, this has the effect of ignoring all but the last instance.

4.2.3.3. Parsing a Key

Given an ASCII string as input_string, return a key. input_string is modified to remove the parsed value.

1. If the first character of input_string is not lcalpha or "*", fail parsing.

2. Let output_string be an empty string.

3. While input_string is not empty:

    1. If the first character of input_string is not one of lcalpha, DIGIT, "_", "-", ".", or "*", return output_string.

    2. Let char be the result of consuming the first character of input_string.

    3. Append char to output_string.

4. Return output_string.

4.2.4. Parsing an Integer or Decimal

Given an ASCII string as input_string, return an Integer or Decimal. input_string is modified to remove the parsed value.

NOTE: This algorithm parses both Integers (Section 3.3.1) and Decimals (Section 3.3.2), and returns the corresponding structure.

1. Let type be "integer".

2. Let sign be 1.

3. Let input_number be an empty string.

4. If the first character of input_string is "-", consume it and set sign to -1.

5. If input_string is empty, there is an empty integer; fail parsing.

6. If the first character of input_string is not a DIGIT, fail parsing.

7. While input_string is not empty:

   1. Let char be the result of consuming the first character of input_string.

   2. If char is a DIGIT, append it to input_number.

   3. Else, if type is "integer" and char is ".":

      1. If input_number contains more than 12 characters, fail parsing.

      2. Otherwise, append char to input_number and set type to "decimal".

   4. Otherwise, prepend char to input_string, and exit the loop.

   5. If type is "integer" and input_number contains more than 15 characters, fail parsing.

   6. If type is "decimal" and input_number contains more than 16 characters, fail parsing.

8. If type is "integer":

   1. Parse input_number as an integer and let output_number be the product of the result and sign.

   2. If output_number is outside the range -999,999,999,999,999 to 999,999,999,999,999 inclusive, fail parsing.

9. Otherwise:

   1. If the final character of input_number is ".", fail parsing.

   2. If the number of characters after "." in input_number is greater than three, fail parsing.

   3. Parse input_number as a decimal number and let output_number be the product of the result and sign.

10. Return output_number.

4.2.5. Parsing a String

Given an ASCII string as input_string, return an unquoted String. input_string is modified to remove the parsed value.

1. Let output_string be an empty string.

2. If the first character of input_string is not DQUOTE, fail parsing.

3. Discard the first character of input_string.

4. While input_string is not empty:

    1. Let char be the result of consuming the first character of input_string.

    2. If char is a backslash ("\"):

        1. If input_string is now empty, fail parsing.

        2. Let next_char be the result of consuming the first character of input_string.

        3. If next_char is not DQUOTE or "\", fail parsing.

        4. Append next_char to output_string.

    3. Else, if char is DQUOTE, return output_string.

    4. Else, if char is in the range %x00-1f or %x7f (i.e., is not in VCHAR or SP), fail parsing.

    5. Else, append char to output_string.

5. Reached the end of input_string without finding a closing DQUOTE; fail parsing.

4.2.6. Parsing a Token

Given an ASCII string as input_string, return a Token. input_string is modified to remove the parsed value.

1. If the first character of input_string is not ALPHA or "*", fail parsing.

2. Let output_string be an empty string.

3. While input_string is not empty:

    1. If the first character of input_string is not in tchar, ":" or "/", return output_string.

    2. Let char be the result of consuming the first character of input_string.

    3. Append char to output_string.

4. Return output_string.

4.2.7. Parsing a Byte Sequence

Given an ASCII string as input_string, return a Byte Sequence. input_string is modified to remove the parsed value.

1. If the first character of input_string is not ":", fail parsing.

2. Discard the first character of input_string.

3. If there is not a ":" character before the end of input_string, fail parsing.

4. Let b64_content be the result of consuming content of input_string up to but not including the first instance of the character ":".

5. Consume the ":" character at the beginning of input_string.

6. If b64_content contains a character not included in ALPHA, DIGIT, "+", "/" and "=", fail parsing.

7. Let binary_content be the result of Base 64 Decoding [RFC4648] b64_content, synthesizing padding if necessary (note the requirements about recipient behavior below).

8. Return binary_content.

Because some implementations of base64 do not allow rejection of encoded data that is not properly "=" padded (see [RFC4648], Section 3.2), parsers SHOULD NOT fail when "=" padding is not present, unless they cannot be configured to do so.

Because some implementations of base64 do not allow rejection of encoded data that has non-zero pad bits (see [RFC4648], Section 3.5), parsers SHOULD NOT fail when non-zero pad bits are present, unless they cannot be configured to do so.

This specification does not relax the requirements in [RFC4648], Section 3.1 and 3.3; therefore, parsers MUST fail on characters outside the base64 alphabet, and on line feeds in encoded data.

4.2.8. Parsing a Boolean

Given an ASCII string as input_string, return a Boolean. input_string is modified to remove the parsed value.

1. If the first character of input_string is not "?", fail parsing.

2. Discard the first character of input_string.

3. If the first character of input_string matches "1", discard the first character, and return true.

4. If the first character of input_string matches "0", discard the first character, and return false.

5. No value has matched; fail parsing.

5. IANA Considerations

This document has no actions for IANA.

6. Security Considerations

The size of most types defined by Structured Fields is not limited; as a result, extremely large fields could be an attack vector (e.g., for resource consumption). Most HTTP implementations limit the sizes of individual fields as well as the overall header or trailer section size to mitigate such attacks.

It is possible for parties with the ability to inject new HTTP fields to change the meaning of a Structured Field. In some circumstances, this will cause parsing to fail, but it is not possible to reliably fail in all such circumstances.

7. References

7.1. Normative References

[RFC0020] Cerf, V., "ASCII format for network interchange", STD 80, RFC 20, DOI 10.17487/RFC0020, October 1969, <https://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, <https://www.rfc-editor.org/info/rfc2119>.

[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>.

[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>.

[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

[IEEE754] IEEE, "IEEE Standard for Floating-Point Arithmetic", IEEE 754-2019, DOI 10.1109/IEEESTD.2019.8766229, ISBN 978-1-5044-5924-2, July 2019, <https://ieeexplore.ieee.org/document/8766229>.

[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>.

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

[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>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[UTF-8]   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/std63>.

7.3. URIs

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

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

[3] https://github.com/httpwg/http-extensions/labels/header-structure

[4] https://github.com/httpwg/structured-field-tests

[5] https://github.com/httpwg/wiki/wiki/Structured-Headers

[6] https://github.com/httpwg/structured-field-tests

Appendix A. Frequently Asked Questions

A.1. Why not JSON?

Earlier proposals for Structured Fields were based upon JSON [RFC8259]. However, constraining its use to make it suitable for HTTP header fields required senders and recipients to implement specific additional handling.

For example, JSON has specification issues around large numbers and objects with duplicate members. Although advice for avoiding these issues is available (e.g., [RFC7493]), it cannot be relied upon.

Likewise, JSON strings are by default Unicode strings, which have a number of potential interoperability issues (e.g., in comparison). Although implementers can be advised to avoid non-ASCII content where unnecessary, this is difficult to enforce.

Another example is JSON's ability to nest content to arbitrary depths. Since the resulting memory commitment might be unsuitable (e.g., in embedded and other limited server deployments), it's necessary to limit it in some fashion; however, existing JSON implementations have no such limits, and even if a limit is specified, it's likely that some field definition will find a need to violate it.

Because of JSON's broad adoption and implementation, it is difficult to impose such additional constraints across all implementations; some deployments would fail to enforce them, thereby harming interoperability. In short, if it looks like JSON, people will be tempted to use a JSON parser / serializer on field values.

Since a major goal for Structured Fields is to improve interoperability and simplify implementation, these concerns led to a format that requires a dedicated parser and serializer.

Additionally, there were widely shared feelings that JSON doesn't "look right" in HTTP fields.

Appendix B. Implementation Notes

A generic implementation of this specification should expose the top-level serialize (Section 4.1) and parse (Section 4.2) functions. They need not be functions; for example, it could be implemented as an object, with methods for each of the different top-level types.

For interoperability, it's important that generic implementations be complete and follow the algorithms closely; see Section 1.1. To aid this, a common test suite is being maintained by the community at https://github.com/httpwg/structured-field-tests [6].

Implementers should note that Dictionaries and Parameters are order-preserving maps. Some fields may not convey meaning in the ordering of these data types, but it should still be exposed so that applications which need to use it will have it available.

Likewise, implementations should note that it's important to preserve the distinction between Tokens and Strings. While most programming languages have native types that map to the other types well, it may be necessary to create a wrapper "token" object or use a parameter on functions to assure that these types remain separate.

The serialization algorithm is defined in a way that it is not strictly limited to the data types defined in Section 3 in every case. For example, Decimals are designed to take broader input and round to allowed values.

Implementations are allowed to limit the allowed size of different structures, subject to the minimums defined for each type. When a structure exceeds an implementation limit, that structure fails parsing or serialisation.

Appendix C. Changes

RFC Editor: Please remove this section before publication.

C.1. Since draft-ietf-httpbis-header-structure-18

- Use "sf-" prefix for ABNF, not "sh-".
- Fix indentation in Dictionary serialisation (#1164).
- Add example for Token; tweak example field names (#1147).
- Editorial improvements.
- Note that exceeding implementation limits implies failure.
- Talk about specifying order of Dictionary members and Parameters, not cardinality.
- Allow (but don't require) parsers to fail when a single field line isn't valid.
- Note that some aspects of Integers and Decimals are not necessarily preserved.
- Allow Lists and Dictionaries to be delimited by OWS, rather than *SP, to make parsing more robust.

C.2. Since draft-ietf-httpbis-header-structure-17

- Editorial improvements.

C.3. Since draft-ietf-httpbis-header-structure-16

- Editorial improvements.
- Discussion on forwards compatibility.

C.4. Since draft-ietf-httpbis-header-structure-15

- Editorial improvements.
- Use HTTP field terminology more consistently, in line with recent changes to HTTP-core.
- String length requirements apply to decoded strings (#1051).
- Correctly round decimals in serialisation (#1043).
- Clarify input to serialisation algorithms (#1055).
- Omitted True dictionary value can have parameters (#1083).

- o Keys can now start with '*' (#1068).

C.5. Since draft-ietf-httpbis-header-structure-14

- o Editorial improvements.
- o Allow empty dictionary values (#992).
- o Change value of omitted parameter value to True (#995).
- o Explain more about splitting dictionaries and lists across header instances (#997).
- o Disallow HTAB, replace OWS with spaces (#998).
- o Change byte sequence delimiters from "*" to ":" (#991).
- o Allow tokens to start with "*" (#991).
- o Change Floats to fixed-precision Decimals (#982).
- o Round the fractional component of decimal, rather than truncating it (#982).
- o Handle duplicate dictionary and parameter keys by overwriting their values, rather than failing (#997).
- o Allow "." in key (#1027).
- o Check first character of key in serialisation (#1037).
- o Talk about greasing headers (#1015).

C.6. Since draft-ietf-httpbis-header-structure-13

- o Editorial improvements.
- o Define "structured header name" and "structured header value" terms (#908).
- o Corrected text about valid characters in strings (#931).
- o Removed most instances of the word "textual", as it was redundant (#915).
- o Allowed parameters on Items and Inner Lists (#907).
- o Expand the range of characters in token (#961).

- Disallow OWS before ";" delimiter in parameters (#961).

C.7. Since draft-ietf-httpbis-header-structure-12

- Editorial improvements.
- Reworked float serialisation (#896).
- Don't add a trailing space in inner-list (#904).

C.8. Since draft-ietf-httpbis-header-structure-11

- Allow * in key (#844).
- Constrain floats to six digits of precision (#848).
- Allow dictionary members to have parameters (#842).

C.9. Since draft-ietf-httpbis-header-structure-10

- Update abstract (#799).
- Input and output are now arrays of bytes (#662).
- Implementations need to preserve difference between token and string (#790).
- Allow empty dictionaries and lists (#781).
- Change parameterized lists to have primary items (#797).
- Allow inner lists in both dictionaries and lists; removes lists of lists (#816).
- Subsume Parameterised Lists into Lists (#839).

C.10. Since draft-ietf-httpbis-header-structure-09

- Changed Boolean from T/F to 1/0 (#784).
- Parameters are now ordered maps (#765).
- Clamp integers to 15 digits (#737).

C.11. Since draft-ietf-httpbis-header-structure-08

- Disallow whitespace before items properly (#703).
- Created "key" for use in dictionaries and parameters, rather than relying on identifier (#702). Identifiers have a separate minimum supported size.
- Expanded the range of special characters allowed in identifier to include all of ALPHA, ".", ":", and "%" (#702).
- Use "?" instead of "!" to indicate a Boolean (#719).
- Added "Intentionally Strict Processing" (#684).
- Gave better names for referring specs to use in Parameterised Lists (#720).
- Added Lists of Lists (#721).
- Rename Identifier to Token (#725).
- Add implementation guidance (#727).

C.12. Since draft-ietf-httpbis-header-structure-07

- Make Dictionaries ordered mappings (#659).
- Changed "binary content" to "byte sequence" to align with Infra specification (#671).
- Changed "mapping" to "map" for #671.
- Don’t fail if byte sequences aren’t "=" padded (#658).
- Add Booleans (#683).
- Allow identifiers in items again (#629).
- Disallowed whitespace before items (#703).
- Explain the consequences of splitting a string across multiple headers (#686).

C.13. Since draft-ietf-httpbis-header-structure-06

- Add a FAQ.
- Allow non-zero pad bits.
- Explicitly check for integers that violate constraints.

C.14. Since draft-ietf-httpbis-header-structure-05

- Reorganise specification to separate parsing out.
- Allow referencing specs to use ABNF.
- Define serialisation algorithms.
- Refine relationship between ABNF, parsing and serialisation algorithms.

C.15. Since draft-ietf-httpbis-header-structure-04

- Remove identifiers from item.
- Remove most limits on sizes.
- Refine number parsing.

C.16. Since draft-ietf-httpbis-header-structure-03

- Strengthen language around failure handling.

C.17. Since draft-ietf-httpbis-header-structure-02

- Split Numbers into Integers and Floats.
- Define number parsing.
- Tighten up binary parsing and give it an explicit end delimiter.
- Clarify that mappings are unordered.
- Allow zero-length strings.
- Improve string parsing algorithm.
- Improve limits in algorithms.
- Require parsers to combine header fields before processing.

- Throw an error on trailing garbage.

C.18. Since draft-ietf-httpbis-header-structure-01

- Replaced with draft-nottingham-structured-headers.

C.19. Since draft-ietf-httpbis-header-structure-00

- Added signed 64bit integer type.
- Drop UTF8, and settle on BCP137 ::EmbeddedUnicodeChar for h1-unicode-string.
- Change h1_blob delimiter to ":" since "'" is valid t_char

Acknowledgements

Many thanks to Matthew Kerwin for his detailed feedback and careful consideration during the development of this specification.

Thanks also to Ian Clelland, Roy Fielding, Anne van Kesteren, Kazuho Oku, Evert Pot, Julian Reschke, Martin Thomson, Mike West, and Jeffrey Yasskin for their contributions.

Authors' Addresses

Mark Nottingham
Fastly

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

Poul-Henning Kamp
The Varnish Cache Project

Email: phk@varnish-cache.org

HTTP M. Bishop
Internet-Draft Akamai
Intended status: Standards Track N. Sullivan
Expires: November 15, 2020 Cloudflare
M. Thomson
Mozilla
May 14, 2020

Secondary Certificate Authentication in HTTP/2
draft-ietf-httpbis-http2-secondary-certs-06

Abstract

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

Note to Readers

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

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

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

HTTP clients need to know that the content they receive on a connection comes from the origin that they intended to retrieve it from. The traditional form of server authentication in HTTP has been in the form of a single X.509 certificate provided during the TLS ([RFC5246], [RFC8446]) handshake.

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

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

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

1.1. Server Certificate Authentication

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

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

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

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

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

1.2. Client Certificate Authentication

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

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

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

1.2.1. HTTP/1.1 Using TLS 1.2 and Earlier

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

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

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

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

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

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

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

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

1.2.2. HTTP/1.1 Using TLS 1.3

TLS 1.3 [RFC8446] introduces a new client authentication mechanism that allows for clients to authenticate after the handshake has been completed. For the purposes of authenticating an HTTP request, this is functionally equivalent to renegotiation. Figure 2 shows the simpler exchange this enables.

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

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

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

1.2.3. HTTP/2

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

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

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

[RFC7540] prohibits renegotiation after any application data has been sent. This completely blocks reactive certificate authentication in HTTP/2 using TLS 1.2. If this restriction were relaxed by an extension or update to HTTP/2, such an identifier could be added to TLS 1.2 by means of an extension to TLS. Unfortunately, many TLS 1.2 implementations do not permit application data to continue during a renegotiation. This is problematic for a multiplexed protocol like HTTP/2.

1.3. HTTP-Layer Certificate Authentication

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

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

TLS Exported Authenticators are structured messages that can be exported by either party of a TLS connection and validated by the other party. Given an established TLS connection, a request can be constructed which describes the desired certificate and an authenticator message can be constructed proving possession of a certificate and a corresponding private key. Both requests and authenticators can be generated by either the client or the server. Exported Authenticators use the message structures from Sections 4.3.2 and 4.4 of [RFC8446], but different parameters.

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

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

Section 2 describes how the feature is employed, defining means to detect support in peers (Section 2.1), make certificates and requests available (Section 2.2), and indicate when streams are blocked waiting on an appropriate certificate (Section 2.3). Section 3 defines the required frame types, which parallel the TLS 1.3 message exchange. Finally, Section 4 defines new error types which can be used to notify peers when the exchange has not been successful.

1.4. Terminology

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

2. Discovering Additional Certificates at the HTTP/2 Layer

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

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

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

2.1. Indicating Support for HTTP-Layer Certificate Authentication

Clients and servers that will accept requests for HTTP-layer certificate authentication indicate this using the HTTP/2 "SETTINGS_HTTP_CLIENT_CERT_AUTH" (0xSETTING-TBD1) and "SETTINGS_HTTP_SERVER_CERT_AUTH" (0xSETTING-TBD2) settings.

The initial value for both settings is 0, indicating that the peer does not support HTTP-layer certificate authentication. If a peer does support HTTP-layer certificate authentication, one or both of the values is non-zero. "SETTINGS_HTTP_CLIENT_CERT_AUTH" indicates that servers can use certificates for client authentication, while "SETTINGS_HTTP_SERVER_CERT_AUTH" indicates that servers are able to offer additional certificates to demonstrate control over other origin hostnames.

In order to ensure that the TLS connection is direct to the server, rather than via a TLS-terminating proxy, each side will separately compute and confirm the value of these settings. The setting values are derived from a TLS exporter (see Section 7.5 of [RFC8446] and [RFC5705] for more details on exporters). Clients MUST NOT use an early exporter during their 0-RTT flight, but MUST send an updated SETTINGS frame using a regular exporter after the TLS handshake completes.

The exporter is constructed with the following input:

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

The value of the exporter is split into two four-byte values. The first four bytes are used for the "SETTINGS_HTTP_CLIENT_CERT_AUTH" value, while the following four bytes are used for the "SETTINGS_HTTP_SERVER_CERT_AUTH" value.

Each is converted to a setting value as:

Exporter fragment | 0x80000000

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

2.2. Making Certificates or Requests Available

When both peers have advertised support for HTTP-layer certificates in a given direction as in Section 2.1, the indicated endpoint can supply additional certificates into the connection at any time. That is, if both endpoints have sent "SETTINGS_HTTP_SERVER_CERT_AUTH" and validated the value received from the peer, the server may send certificates. If both endpoints have sent "SETTINGS_HTTP_CLIENT_CERT_AUTH" and validated the value received from the peer, the client may send certificates.

Implementations which predict a certificate will be required could supply the certificate before being asked. These certificates are available for reference by future "USE_CERTIFICATE" frames.

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

Certificates supplied by clients MUST NOT be considered by servers when processing a request unless the client explicitly authorizes their use. Clients MAY send "USE_CERTIFICATE" frame with the "UNSOLICITED" flag set to indicate that an available certificate should be considered on a new request.

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

Figure 3: Proactive server authentication

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

Figure 4: Proactive client authentication

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

2.3. Requiring Certificate Authentication

2.3.1. Requiring Additional Server Certificates

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

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

If the server does not have the desired certificate, it MUST send an Empty Authenticator, as described in Section 5 of [I-D.ietf-tls-exported-authenticator], in a "CERTIFICATE" frame in response to the request, followed by a "USE_CERTIFICATE" frame for stream zero which references the Empty Authenticator.

If a server has not advertised support for HTTP-layer certificates, fails to provide a requested certificate, or provides a certificate which is unacceptable to the client, the client MUST NOT send any requests for resources in that origin on the current connection. The client MAY open a new connection in an effort to reach an authoritative server.

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

Figure 5: Client-requested certificate

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

2.3.2. Requiring Additional Client Certificates

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

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

Figure 6: Reactive certificate authentication

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

If the client has not advertised support for HTTP-layer certificates, fails to provide a requested certificate, or provides a certificate the server is unable to verify, the server processes the request based solely on the certificate provided during the TLS handshake, if any. This might result in an error response via HTTP, such as a status code 403 (Not Authorized).

3. Certificates Frames for HTTP/2

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

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

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

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

Figure 7: Frame correlation

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

3.1. The CERTIFICATE_NEEDED Frame

The "CERTIFICATE_NEEDED" frame (0xFRAME-TBD1) is sent on stream zero to indicate that the HTTP request on the indicated stream is blocked pending certificate authentication. The frame includes stream ID and a request identifier which can be used to correlate the stream with a previous "CERTIFICATE_REQUEST" frame sent on stream zero. The "CERTIFICATE_REQUEST" describes the certificate the sender requires to make progress on the stream in question.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-------------------------------------------------------------+
|R|                        Stream ID (31)                       |
+-+-----------------------------+-------------------------------+
|        Request-ID (16)        |
+-------------------------------+
```

Figure 8: CERTIFICATE_NEEDED frame payload

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

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

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

The "CERTIFICATE_NEEDED" frame MUST NOT be sent to a client which has not advertised the "SETTINGS_HTTP_CLIENT_CERT_AUTH", or to a server which has not advertised the "SETTINGS_HTTP_SERVER_CERT_AUTH" setting. An endpoint which receives a "CERTIFICATE_NEEDED" frame but did not advertise support MAY treat this as a connection error of type "CERTIFICATE_WITHOUT_CONSENT".

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

3.2. The USE_CERTIFICATE Frame

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

The "USE_CERTIFICATE" frame defines a single flag:

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

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-------------------------------------------------------------+
|R|                        Stream ID (31)                       |
+-+-----------------------------+-------------------------------+
|         [Cert-ID (16)]        |
+-------------------------------+
```

Figure 9: USE_CERTIFICATE frame payload

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

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

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

Receipt of more than one unsolicited "USE_CERTIFICATE" frame or an unsolicited "USE_CERTIFICATE" frame which is not the first in reference to a given stream MUST be treated as a stream error of type "CERTIFICATE_OVERUSED".

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

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

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

3.3. The CERTIFICATE_REQUEST Frame

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

The "CERTIFICATE_REQUEST" frame SHOULD NOT be sent to a client which has not advertised the "SETTINGS_HTTP_CLIENT_CERT_AUTH", or to a server which has not advertised the "SETTINGS_HTTP_SERVER_CERT_AUTH" setting.

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

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-------------------------------+-------------------------------+
|        Request-ID (16)        |          Request (?)        ...
+---------------------------------------------------------------+
```

Figure 10: CERTIFICATE_REQUEST frame payload

The frame contains the following fields:

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

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

3.3.1. Exported Authenticator Request Characteristics

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

Upon receipt of a "CERTIFICATE_REQUEST" frame, the recipient MUST verify that the first two octets of the authenticator's "certificate_request_context" matches the Request-ID presented in the frame.

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

3.4. The CERTIFICATE Frame

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

The "CERTIFICATE" frame defines two flags:

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

UNSOLICITED (0x02): Indicates that the exported authenticator does not contain a Request-ID.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-------------------------------+-------------------------------+
|          Cert-ID (16)         |        Request-ID (16)        |
+-------------------------------+-------------------------------+
|                   Authenticator Fragment (*)                ...
+---------------------------------------------------------------+
```

Figure 11: CERTIFICATE frame payload

The frame contains the following fields:

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

Request-ID: "Request-ID" is an optional 16-bit opaque identifier used to correlate this exported authenticator with the request which triggered it, if any. This field is present only if the "UNSOLICITED" flag is not set.

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

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

If the "UNSOLICITED" flag is not set, the "CERTIFICATE" frame also contains a Request-ID indicating the certificate request which caused this exported authenticator to be generated. The value of this flag and the contents of the Request-ID field MUST NOT differ between frames with the same Cert-ID.

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

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

3.4.1. Exported Authenticator Characteristics

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

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

- Verify that either the "UNSOLICITED" flag is set (clients only) or that the Request-ID field contains the Request-ID of a previously-sent "CERTIFICATE_REQUEST" frame.

- Using the "get context" API, retrieve the "certificate_request_context" used to generate the authenticator, if any. Verify that the "certificate_request_context" begins with the supplied Request-ID, if any.

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

If the authenticator cannot be validated, this SHOULD be treated as a connection error of type "CERTIFICATE_UNREADABLE".

Once the authenticator is accepted, the endpoint can perform any other checks for the acceptability of the certificate itself. Clients MUST NOT accept any end-entity certificate from an exported authenticator which does not contain the Required Domain extension; see Section 5 and Section 6.1.

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.

There are two classes of errors which might be encountered, and they are handled differently.

4.1. Misbehavior

This category of errors could indicate a peer failing to follow restrictions in this document, or might indicate that the connection is not fully secure. These errors are fatal to stream or connection, as appropriate.

CERTIFICATE_OVERUSED (0xERROR-TBD1): More certificates were used on a request than were requested

CERTIFICATE_WITHOUT_CONSENT (0xERROR-TBD2): A CERTIFICATE_NEEDED frame was received by a peer which did not indicate support for this extension.

CERTIFICATE_UNREADABLE (0xERROR-TBD3): An exported authenticator could not be validated.

4.2. Invalid Certificates

Unacceptable certificates (expired, revoked, or insufficient to satisfy the request) are not treated as stream or connection errors. This is typically not an indication of a protocol failure. Servers SHOULD process requests with the indicated certificate, likely resulting in a "4XX"-series status code in the response. Clients SHOULD establish a new connection in an attempt to reach an authoritative server.

5. Required Domain Certificate Extension

The Required Domain extension allows certificates to limit their use with Secondary Certificate Authentication. A client MUST verify that the server has proven ownership of the indicated identity before accepting the limited certificate over Secondary Certificate Authentication.

The identity in this extension is a restriction asserted by the requester of the certificate and is not verified by the CA. Conforming CAs SHOULD mark the requiredDomain extension as non-critical. Conforming CAs MUST require the presence of a CAA record [RFC6844] prior to issuing a certificate with this extension. Because a Required Domain value of "*" has a much higher risk of reuse if compromised, conforming Certificate Authorities are encouraged to require more extensive verification prior to issuing such a certificate.

The required domain is represented as a GeneralName, as specified in Section 4.2.1.6 of [RFC5280]. Unlike the subject field, conforming CAs MUST NOT issue certificates with a requiredDomain extension containing empty GeneralName fields. Clients that encounter such a certificate when processing a certification path MUST consider the certificate invalid.

The wildcard character "_" MAY be used to represent that any previously authenticated identity is acceptable. This character MUST be the entirety of the name if used and MUST have a type of "dNSName". (That is, "_" is acceptable, but "_.com" and "w_.example.com" are not).

```
id-ce-requiredDomain OBJECT IDENTIFIER ::=  { id-ce TBD1 }

RequiredDomain ::= GeneralName
```

6. 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.

6.1. 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. Clients SHOULD consult DNS for hostnames presented in secondary certificates if they would have done so for the same hostname if it were present in the primary certificate.

As recommended in [RFC8336], clients opting not to consult DNS ought to employ some alternative means to increase confidence that the certificate is legitimate.

One such means is the Required Domain certificate extension defined in {extension}. Clients MUST require that server certificates presented via this mechanism contain the Required Domain extension and require that a certificate previously accepted on the connection (including the certificate presented in TLS) lists the Required Domain in the Subject field or the Subject Alternative Name extension.

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.

6.2. 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.

Servers can also learn information about clients using this mechanism. The hostnames a user agent finds interesting and retrieves certificates for might indicate origins the user has previously accessed.

6.3. Denial of Service

Failure to provide a certificate for 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.

6.4. Persistence of Service

CNAME records in the DNS are frequently used to delegate authority for an origin to a third-party provider. This delegation can be changed without notice, even to the third-party provider, simply by modifying the CNAME record in question.

After the owner of the domain has redirected traffic elsewhere by changing the CNAME, new connections will not arrive for that origin, but connections which are properly directed to this provider for other origins would continue to claim control of this origin (via ORIGIN frame and Secondary Certificates). This is proper behavior based on the third-party provider's configuration, but would likely not be what is intended by the owner of the origin.

This is not an issue which can be mitigated by the protocol, but something about which third-party providers SHOULD educate their customers before using the features described in this document.

6.5. 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.

7. IANA Considerations

This draft adds entries in three registries.

The feature negotiation settings is registered in Section 7.1. Four frame types are registered in Section 7.2. Six error codes are registered in Section 7.3.

7.1. HTTP/2 Settings

The SETTINGS_HTTP_CLIENT_CERT_AUTH and SETTINGS_HTTP_SERVER_CERT_AUTH settings are registered in the "HTTP/2 Settings" registry established in [RFC7540].

Name	Code	Initial Value	Specification
HTTP_CLIENT_CERT_AUTH	0xSETTING-TBD1	0	Section 2.1
HTTP_SERVER_CERT_AUTH	0xSETTING-TBD2	0	Section 2.1

7.2. 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

7.3. New HTTP/2 Error Codes

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

Name	Code	Specification
CERTIFICATE_OVERUSED	0xERROR-TBD1	Section 4
CERTIFICATE_WITHOUT_CONSENT	0xERROR-TBD2	Section 4
CERTIFICATE_UNREADABLE	0xERROR-TBD3	Section 4

8. References

8.1. Normative References

[I-D.ietf-tls-exported-authenticator]
Sullivan, N., "Exported Authenticators in TLS", draft-ietf-tls-exported-authenticator-11 (work in progress), December 2019.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

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

[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>.

[RFC6844] Hallam-Baker, P. and R. Stradling, "DNS Certification Authority Authorization (CAA) Resource Record", RFC 6844, DOI 10.17487/RFC6844, January 2013, <https://www.rfc-editor.org/info/rfc6844>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

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

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

8.2. Informative References

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

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

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

8.3. URIs

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

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

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

Appendix A. Change Log

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

A.1. Since draft-ietf-httpbis-http2-secondary-certs-04:

Editorial updates only.

A.2. Since draft-ietf-httpbis-http2-secondary-certs-03:

- "CERTIFICATE_REQUEST" frames contain the Request-ID, which MUST be checked against the "certificate_request_context" of the Exported Authenticator Request

- "CERTIFICATE" frames contain the Request-ID to which they respond, unless the UNSOLICITED flag is set

- The Required Domain extension is defined for certificates, which must be present for certificates presented by servers

A.3. Since draft-ietf-httpbis-http2-secondary-certs-02:

Editorial updates only.

A.4. Since draft-ietf-httpbis-http2-secondary-certs-01:

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

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

A.5. Since draft-ietf-httpbis-http2-secondary-certs-00:

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

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

A.6. Since draft-bishop-httpbis-http2-additional-certs-05:

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

Acknowledgements

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

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

Authors' Addresses

Mike Bishop
Akamai

Email: mbishop@evequefou.be

Nick Sullivan
Cloudflare

Email: nick@cloudflare.com

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

HTTP C. Pratt
Internet-Draft
Intended status: Experimental D. Thakore
Expires: September 7, 2019 CableLabs
B. Stark
AT&T
March 6, 2019

HTTP Random Access and Live Content
draft-ietf-httpbis-rand-access-live-04

Abstract

To accommodate byte range requests for content that has data appended over time, this document defines semantics that allow a HTTP client and server to perform byte-range GET and HEAD requests that start at an arbitrary byte offset within the representation and ends at an indeterminate offset.

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/labels/rand-access-live>.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Some Hypertext Transfer Protocol (HTTP) clients use byte-range requests (Range requests using the "bytes" Range Unit) to transfer select portions of large representations ([RFC7233]). And in some cases large representations require content to be continuously or periodically appended - such as representations consisting of live audio or video sources, blockchain databases, and log files. Clients cannot access the appended/live content using a Range request with the bytes range unit using the currently defined byte-range semantics without accepting performance or behavior sacrifices which are not acceptable for many applications.

For instance, HTTP clients have the ability to access appended content on an indeterminate-length resource by transferring the entire representation from the beginning and continuing to read the appended content as it's made available. Obviously, this is highly inefficient for cases where the representation is large and only the most recently appended content is needed by the client.

Alternatively, clients can also access appended content by sending periodic open-ended bytes Range requests using the last-known end byte position as the range start. Performing low-frequency periodic bytes Range requests in this fashion (polling) introduces latency since the client will necessarily be somewhat behind the aggregated content - mimicking the behavior (and latency) of segmented content representations such as "HTTP Live Streaming" (HLS, [RFC8216]) or "Dynamic Adaptive Streaming over HTTP" (MPEG-DASH, [DASH]). And while performing these Range requests at higher frequency can reduce this latency, it also incurs more processing overhead and HTTP exchanges as many of the requests will return no content - since content is usually aggregated in groups of bytes (e.g. a video frame, audio sample, block, or log entry).

This document describes a usage model for range requests which enables efficient retrieval of representations that are appended to over time by using large values and associated semantics for communicating range end positions. This model allows representations to be progressively delivered by servers as new content is added. It also ensures compatibility with servers and intermediaries that don't support this technique.

1.1. Requirements Language

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].

1.2. Notational Conventions

This document cites productions in Augmented Backus-Naur Form (ABNF) productions from [RFC7233], using the notation defined in [RFC5234].

2. Performing Range requests on Random-Access Aggregating ("live") Content

This document recommends a two-step process for accessing resources that have indeterminate length representations.

Two steps are necessary because of limitations with the Range request header fields and the Content-Range response header fields. A server cannot know from a range request that a client wishes to receive a response that does not have a definite end. More critically, the header fields do not allow the server to signal that a resource has indeterminate length without also providing a fixed portion of the resource.

A client first learns that the resource has a representation of indeterminate length by requesting a range of the resource. The server responds with the range that is available, but indicates that the length of the representation is unknown using the existing Content-Range syntax. See Section 2.1 for details and examples.

Once the client knows the resource has indeterminate length, it can request a range with a very large end position from the resource. The client chooses an explicit end value larger than can be transferred in the foreseeable term. A server which supports range requests of indeterminate length signals its understanding of the client's indeterminate range request by indicating that the range it is providing has a range end that exactly matches the client's requested range end rather than a range that is bounded by what is currently available. See Section 2.2 for details.

2.1. Establishing the Randomly Accessible Byte Range

Establishing if a representation is continuously aggregating ("live") and determining the randomly-accessible byte range can both be determined using the existing definition for an open-ended byte-range request. Specifically, [RFC7233] defines a byte-range request of the form:

```
byte-range-spec = first-byte-pos "-" [ last-byte-pos ]
```

which allows a client to send a HEAD request with a first-byte-pos and leave last-byte-pos absent. A server that receives a satisfiable byte-range request (with first-byte-pos smaller than the current representation length) may respond with a 206 status code (Partial Content) with a Content-Range header field indicating the currently satisfiable byte range. For example:

```
HEAD /resource HTTP/1.1
Host: example.com
Range: bytes=0-
```

returns a response of the form:

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 0-1234567/*
```

from the server indicating that (1) the complete representation length is unknown (via the "*" in place of the complete-length field) and (2) that only bytes 0-1234567 were accessible at the time the request was processed by the server. The client can infer from this response that bytes 0-1234567 of the representation can be requested and returned in a timely fashion (the bytes are immediately available).

2.2. Byte-Range Requests Beyond the Randomly Accessible Byte Range

Once a client has determined that a representation has an indeterminate length and established the byte range that can be accessed, it may want to perform a request with a start position within the randomly-accessible content range and an end position at an indefinite "live" point - a point where the byte-range GET request is fulfilled on-demand as the content is aggregated.

For example, for a large video asset, a client may wish to start a content transfer from the video "key" frame immediately before the point of aggregation and continue the content transfer indefinitely as content is aggregated - in order to support low-latency startup of a live video stream.

Unlike a byte-range Range request, a byte-range Content-Range response header field cannot be "open ended", per [RFC7233]:

```
byte-content-range  = bytes-unit SP
                       ( byte-range-resp / unsatisfied-range )

byte-range-resp     = byte-range "/" ( complete-length / "*" )
byte-range          = first-byte-pos "-" last-byte-pos
unsatisfied-range   = "*/" complete-length

complete-length     = 1*DIGIT
```

Specifically, last-byte-pos is required in byte-range. So in order to preserve interoperability with existing HTTP clients, servers, proxies, and caches, this document proposes a mechanism for a client to indicate support for handling an indeterminate-length byte-range response, and a mechanism for a server to indicate if/when it’s providing an indeterminate-length response.

A client can indicate support for handling indeterminate-length byte-range responses by providing a very large value for the last-byte-pos in the byte-range request. For example, a client can perform a byte-range GET request of the form:

```
GET /resource HTTP/1.1
Host: example.com
Range: bytes=1230000-999999999999
```

where the last-byte-pos in the Request is much larger than the last-byte-pos returned in response to an open-ended byte-range HEAD request, as described above, and much larger than the expected maximum size of the representation. See Section 6 for range value considerations.

In response, a server may indicate that it is supplying a continuously aggregating ("live") response by supplying the client request’s last-byte-pos in the Content-Range response header field.

For example:

```
GET /resource HTTP/1.1
Host: example.com
Range: bytes=1230000-999999999999
```

returns

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 1230000-999999999999/*
```

from the server to indicate that the response will start at byte 1230000 and continues indefinitely to include all aggregated content, as it becomes available.

A server that doesn’t support or supply a continuously aggregating ("live") response will supply the currently satisfiable byte range, as it would with an open-ended byte request.

For example:

```
GET /resource HTTP/1.1
Host: example.com
Range: bytes=1230000-999999999999
```

will return

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 1230000-1234567/*
```

from the server to indicate that the response will start at byte 1230000 and end at byte 1234567 and will not include any aggregated content. This is the response expected from a typical HTTP server - one that doesn't support byte-range requests on aggregating content.

A client that doesn't receive a response indicating it is continuously aggregating must use other means to access aggregated content (e.g. periodic byte-range polling).

A server that does return a continuously aggregating ("live") response should return data using chunked transfer coding and not provide a Content-Length header field. A 0-length chunk indicates the end of the transfer, per [RFC7230].

3. Other Applications of Random-Access Aggregating Content

3.1. Requests Starting at the Aggregation ("Live") Point

A client that wishes to only receive newly-aggregated portions of a resource (i.e., start at the "live" point), can use a HEAD request to learn what range the server has currently available and initiate an indeterminate-length transfer. For example:

```
HEAD /resource HTTP/1.1
Host: example.com
Range: bytes=0-
```

With the Content-Range response header field indicating the range (or ranges) available. For example:

```
206 Partial Content
Content-Range: bytes 0-1234567/*
```

The client can then issue a request for a range starting at the end value (using a very large value for the end of a range) and receive only new content.

```
GET /resource HTTP/1.1
Host: example.com
Range: bytes=1234567-999999999999
```

with a server returning a Content-Range response indicating that an indeterminate-length response body will be provided

```
206 Partial Content
Content-Range: bytes 1234567-999999999999/*
```

3.2. Shift Buffer Representations

Some representations lend themselves to front-end content removal in addition to aggregation. While still supporting random access, representations of this type have a portion at the beginning (the "0" end) of the randomly-accessible region that become inaccessible over time. Examples of this kind of representation would be an audio-video time-shift buffer or a rolling log file.

For example a Range request containing:

```
HEAD /resource HTTP/1.1
Host: example.com
Range: bytes=0-
```

returns

```
206 Partial Content
Content-Range: bytes 1000000-1234567/*
```

indicating that the first 1000000 bytes were not accessible at the time the HEAD request was processed. Subsequent HEAD requests could return:

```
Content-Range: bytes 1000000-1234567/*

Content-Range: bytes 1010000-1244567/*

Content-Range: bytes 1020000-1254567/*
```

Note though that the difference between the first-byte-pos and last-byte-pos need not be constant.

The client could then follow-up with a GET Range request containing

```
GET /resource HTTP/1.1
Host: example.com
Range: bytes=1020000-999999999999
```

with the server returning

```
206 Partial Content
Content-Range: bytes 1020000-999999999999/*
```

with the response body returning bytes 1020000-1254567 immediately and aggregated ("live") data being returned as the content is aggregated.

A server that doesn't support or supply a continuously aggregating ("live") response will supply the currently satisfiable byte range, as it would with an open-ended byte request.

For example:

```
GET /resource HTTP/1.1
Host: example.com
Range: bytes=0-999999999999
```

will return

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 1020000-1254567/*
```

from the server to indicate that the response will start at byte 1020000, end at byte 1254567, and will not include any aggregated content. This is the response expected from a typical HTTP server - one that doesn't support byte-range requests on aggregating content.

Note that responses to GET requests against shift-buffer representations using Range can be cached by intermediaries, since the Content-Range response header indicates which portion of the representation is being returned in the response body. However GET requests without a Range header cannot be cached since the first byte of the response body can vary from request to request. To ensure Range-less GET requests against shift-buffer representations are not cached, servers hosting a shift-buffer representation should either not return a 200-level response (e.g. sending a 300-level redirect response with a URI that represents the current start of the shift-buffer) or indicate the response is non-cacheable. See HTTP Caching ([RFC7234]) for details on HTTP cache control.

4. Recommendations for Very Large Values

While it would be ideal to define a single numerical Very Large Value, there's no single value that would work for all applications and platforms. e.g. JavaScript numbers cannot represent all integer values above 2^^53, so a JavaScript application may want to use 2^^53-1 for a Very Large Value. This value, however, would not be sufficient for all applications, such as continuously-streaming high-bitrate streams. So the value 2^^53-1 (9007199254740991) is recommended as a Very Large Value unless an application has a good justification to use a smaller or larger value. e.g. If it's always known that the resource won't exceed a value smaller than the recommended Very Large Value for an application, a smaller value can be used. And if it's likely that an application will utilize resources larger than the recommended Very Large Value - such as a continuously aggregating high-bitrate media stream - a larger value should be used.

Note that, in accordance with the semantics defined above, servers that support random-access live content will need to return the last-byte-pos provided in the Range request in some cases - even if the last-byte-pos cannot be represented as a numerical value internally by the server. As is the case with any live/continuously aggregating resource, the server should terminate the content transfer when the end of the resource is reached - whether the end is due to termination of the content source or the content length exceeds the server's maximum representation length.

5. IANA Considerations

This document has no actions for IANA.

6. Security Considerations

As described above, servers need to be prepared to receive last-byte-pos values in Range requests that are numerically larger than the server implementation supports - and return these values in Content-Range response header fields. Servers should check the last-byte-pos value before converting and storing them into numeric form to ensure the value doesn't cause an overflow or index incorrect data. The simplest way to satisfy the live-range semantics defined in this document without potential overflow issues is to store the last-byte-pos as a string value and return it in the byte-range Content-Range response header's last-byte-pos field.

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>.

[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>.

[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, <https://www.rfc-editor.org/info/rfc7233>.

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

7.2. Informative References

[DASH] ISO, "Information technology -- Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment formats", ISO/IEC 23009-1:2014, May 2014, <http://standards.iso.org/ittf/PubliclyAvailableStandards/c065274_ISO_IEC_23009-1_2014.zip>.

[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>.

[RFC8216] Pantos, R., Ed. and W. May, "HTTP Live Streaming", RFC 8216, DOI 10.17487/RFC8216, August 2017, <https://www.rfc-editor.org/info/rfc8216>.

Appendix A. Acknowledgements

Mark Nottingham, Patrick McManus, Julian Reschke, Remy Lebeau, Rodger Combs, Thorsten Lohmar, Martin Thompson, Adrien de Croy, K. Morgan, Roy T. Fielding, Jeremy Poulter.

Authors' Addresses

Craig Pratt
Portland, OR 97229
US

Email: pratt@acm.org

Darshak Thakore
CableLabs
858 Coal Creek Circle
Louisville, CO 80027
US

Email: d.thakore@cablelabs.com

Barbara Stark
AT&T
Atlanta, GA
US

Email: barbara.stark@att.com

HTTP   M. West, Ed.
Internet-Draft   Google, Inc
Obsoletes: 6265 (if approved)   J. Wilander, Ed.
Intended status: Standards Track   Apple, Inc
Expires: October 22, 2020   April 20, 2020

Cookies: HTTP State Management Mechanism
draft-ietf-httpbis-rfc6265bis-06

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.

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/6265bis [3].

Status of This Memo

Copyright Notice

Table of Contents

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.

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 headers 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 3.2.3 of [RFC7230].

2.3. Terminology

The terms "user agent", "client", "server", "proxy", and "origin server" have the same meaning as in the HTTP/1.1 specification ([RFC7230], Section 2).

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 "request-target" as defined in Section 5.3 of [RFC7230].

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", "WorkerGlobalScope", "sandboxed origin browsing context flag", "parent browsing context", "shared worker", "the worker's Documents", "nested browsing context", and "top-level browsing context" are defined in [HTML].

"Service Workers" are defined in the Service Workers specification [SERVICE-WORKERS].

The term "origin", the mechanism of deriving an origin from a URI, and the "the same" matching algorithm for origins are defined in [RFC6454].

"Safe" HTTP methods include "GET", "HEAD", "OPTIONS", and "TRACE", as defined in Section 4.2.1 of [RFC7231].

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", and "target browsing context", are defined in [FETCH].

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

Origin servers MAY send a Set-Cookie response header with any response. User agents MAY ignore Set-Cookie headers contained in responses with 100-level status codes but MUST process Set-Cookie headers contained in other responses (including responses with 400- and 500-level status codes). 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 3.2.2 of [RFC7230]) 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.

3.1. Examples

Using the Set-Cookie header, 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 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 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 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 headers.

4.1. Set-Cookie

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

4.1.1. Syntax

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

```
set-cookie-header = "Set-Cookie:" SP BWS 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
                      ; US-ASCII characters 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 [RFC7231], Section 7.1.1.1>
max-age-av        = "Max-Age" BWS "=" BWS non-zero-digit *DIGIT
                      ; In practice, both expires-av and max-age-av
                      ; are limited to dates representable by the
                      ; user agent.
non-zero-digit    = %x31-39
                      ; digits 1 through 9
domain-av         = "Domain" BWS "=" BWS domain-value
domain-value      = <subdomain>
                      ; defined in [RFC1034], Section 3.5, as
                      ; enhanced by [RFC1123], Section 2.1
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].

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 headers 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 5.4 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 5.3 for how user agents handle this case.)

If a server sends multiple responses containing Set-Cookie headers 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. 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, 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.

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.

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.

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 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 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 5.4 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]).

Although seemingly useful for protecting cookies from active network attackers, the Secure attribute protects only the cookie’s confidentiality. An active network attacker can overwrite Secure cookies from an insecure channel, disrupting their integrity (see Section 8.6 for more details).

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 (such as a web browser API that exposes cookies to scripts).

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 "site.example".

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 5.3.7.1. 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 treated as "None".

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 5.4.

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 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 would be accepted:

```
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 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. 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 that conforms to the following grammar:

```
cookie-header = "Cookie:" SP 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.

Notice that the cookie attributes are not returned. In particular, the server cannot determine from the Cookie header 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 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, servers SHOULD NOT rely upon the serialization order. In particular, if the Cookie header 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.

5. User Agent Requirements

This section specifies the Cookie and Set-Cookie headers 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., for the sake of improved security); however, experiments have shown that such strictness reduces 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 headers.

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:

o 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.)

o 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). For example, if the request-uri contains just a path (and optional query string), then the uri-path is that path (without the %x3F ("?") character or query string), and if the request-uri contains a full absoluteURI, the uri-path is the path component of that URI.

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

A request is "same-site" if its target's URI's origin's registrable domain is an exact match for the request's client's "site for cookies", or if the request has no client. The request is otherwise "cross-site".

For a given request ("request"), the following algorithm returns "same-site" or "cross-site":

1. If "request"'s client is "null", return "same-site".

   Note that this is the case for navigation triggered by the user directly (e.g. by typing directly into a user agent's address bar).

2. Let "site" be "request"'s client's "site for cookies" (as defined in the following sections).

3. Let "target" be the registrable domain of "request"'s current url.

4. If "site" is an exact match for "target", return "same-site".

5. Return "cross-site".

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 registrable domain of that URI's origin represents the context in which a user most likely believes themselves to be interacting. We'll label this domain the "top-level site".

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

For documents which are displayed in nested browsing contexts, we need to audit the origins of each of a document's ancestor browsing contexts' active documents in order to account for the "multiple-nested scenarios" described in Section 4 of [RFC7034]. A document's "site for cookies" is the top-level site if and only if the document and each of its ancestor documents' origins have the same registrable domain as the top-level site. Otherwise its "site for cookies" is the empty string.

Given a Document ("document"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Let "top-document" be the active document in "document"'s browsing context's top-level browsing context.

2. Let "top-origin" be the origin of "top-document"'s URI if "top-document"'s sandboxed origin browsing context flag is set, and "top-document"'s origin otherwise.

3. Let "documents" be a list containing "document" and each of "document"'s ancestor browsing contexts' active documents.

4. For each "item" in "documents":

    1. Let "origin" be the origin of "item"'s URI if "item"'s sandboxed origin browsing context flag is set, and "item"'s origin otherwise.

    2. If "origin"'s host's registrable domain is not an exact match for "top-origin"'s host's registrable domain, return the empty string.

5. Return "top-origin"'s host's registrable domain.

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 cookie" values, the worker's "site for cookies" will be the empty string in cases where the values diverge, and the shared value in cases where the values agree.

Given a WorkerGlobalScope ("worker"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Let "site" be "worker"'s origin's host's registrable domain.

2. For each "document" in "worker"'s Documents:

    1. Let "document-site" be "document"'s "site for cookies" (as defined in Section 5.2.1).

    2. If "document-site" is not an exact match for "site", return the empty string.

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 registrable domain of the Service Worker's URI.

Given a ServiceWorkerGlobalScope ("worker"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Return "worker"'s origin's host's registrable domain.

5.3. The Set-Cookie Header

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. For example, the user agent might wish to block responses to "third-party" requests from setting cookies (see Section 7.1).

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. User agents use this algorithm so as to interoperate with servers that do not follow the recommendations in Section 4.

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 %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.

2. 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.

3. Remove any leading or trailing WSP characters from the name string and the value string.

4. If both the name string and the value string are empty, ignore the set-cookie-string entirely.

5. 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. Process the attribute-name and attribute-value according to the requirements in the following subsections. (Notice that attributes with unrecognized attribute-names are ignored.)

7. 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 5.4 for additional requirements triggered by receiving a cookie.)

5.3.1. 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. If the expiry-time is later than the last date the user agent can represent, the user agent MAY replace the expiry-time with the last representable date.

4. 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.

5. Append an attribute to the cookie-attribute-list with an attribute-name of Expires and an attribute-value of expiry-time.

5.3.2. 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. 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.

5. Append an attribute to the cookie-attribute-list with an attribute-name of Max-Age and an attribute-value of expiry-time.

5.3.3. The Domain Attribute

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

1. If the attribute-value is empty, the behavior is undefined. However, the user agent SHOULD ignore the cookie-av entirely.

2. If the first character of the attribute-value string is %x2E ("."):

    1. Let cookie-domain be the attribute-value without the leading %x2E (".") character.

    Otherwise:

    1. Let cookie-domain be the entire attribute-value.

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.

5.3.4. 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.

5.3.5. 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.

5.3.6. 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.

5.3.7. 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 "None".

2. If cookie-av's attribute-value is a case-insensitive match for "Strict", set "enforcement" to "Strict".

3. If cookie-av's attribute-value is a case-insensitive match for "Lax", set "enforcement" to "Lax".

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

Note: This algorithm maps the "None" value, as well as any unknown value, to the "None" behavior, which is helpful for backwards compatibility when introducing new variants.

5.3.7.1. "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 [RFC7231] sense) HTTP method.

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. 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. For example, the user agent might wish to block receiving cookies

from "third-party" responses or the user agent might not wish to store cookies that exceed some size.

2. 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.

3. 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.

4. 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 an attribute-name of "Domain".

    Otherwise:

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

5. 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. Ignore the cookie entirely and abort these steps.

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

6. If the domain-attribute is non-empty:

    1. If the canonicalized request-host does not domain-match the
        domain-attribute:

        1. Ignore the cookie entirely and abort these steps.

        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.

7. 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 an attribute-name of "Path". Otherwise, set the cookie's path to the default-path of the request-uri.

8. 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.

9. 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.

10. 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.

11. 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.

12. 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'.

13. If the cookie-attribute-list contains an attribute with an attribute-name of "SameSite", 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" (i.e. either "Strict", "Lax", or "None"). Otherwise, set the cookie's same-site-flag to "None".

14. 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 context whose "site for cookies" is not an exact match for request-uri's host's registrable domain, then abort these steps and ignore the newly created cookie entirely.

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 browing 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.

15. 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.

16. 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 "/".

17. 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.

18. 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.

5.5. The Cookie Header

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

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 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 send the cookie-string (defined below) as the value of the header field.

The user agent MUST use an algorithm equivalent to the following algorithm to compute the cookie-string from a cookie store and a request-uri:

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 request-host is identical to the cookie's domain.

     Or:

     + The cookie's host-only-flag is false and the canonicalized request-host domain-matches the cookie's domain.

   * The request-uri's path path-matches the cookie's path.

   * If the cookie's secure-only-flag is true, then the request-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 cookie-string is being generated for a "non-HTTP" API (as defined by the user agent).

   * If the cookie's same-site-flag is not "None", and the HTTP request is cross-site (as defined in Section 5.2) then exclude the cookie unless all of the following statements hold:

     1. The same-site-flag is "Lax"

     2. The HTTP request's method is "safe".

     3. The HTTP request's target browsing context is a top-level browsing context.

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:

o At least 4096 bytes per cookie (as measured by the sum of the length of the cookie's name, value, and attributes).

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

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 being included in every request.

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

6.2. Application Programming Interfaces

One reason the Cookie and Set-Cookie headers 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 headers, 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 are often criticized for letting servers track users. For example, a number of "web analytics" companies use cookies to recognize when a user returns to a web site or visits another web site. Although cookies are not the only mechanism servers can use to track users across HTTP requests, cookies facilitate tracking because they are persistent across user agent sessions and can be shared between hosts.

7.1. Third-Party Cookies

Particularly worrisome are so-called "third-party" cookies. In rendering an HTML document, a user agent often requests resources from other servers (such as advertising networks). These third-party servers can use cookies to track the user even if the user never visits the server directly. For example, if a user visits a site that contains content from a third party and then later visits another site that contains content from the same third party, the third party can track the user between the two sites.

Given this risk to user privacy, some user agents restrict how third-party cookies behave, and those restrictions vary widly. For instance, user agents might block third-party cookies entirely by refusing to send Cookie headers or process Set-Cookie headers during third-party requests. They might take a less draconian approach by partitioning cookies based on the first-party context, sending one set of cookies to a given third party in one first-party context, and another to the same third party in another.

This document grants user agents wide latitude to experiment with third-party cookie policies that balance the privacy and compatibility needs of their users. However, this document does not endorse any particular third-party cookie policy.

Third-party cookie blocking policies are often ineffective at achieving their privacy goals if servers attempt to work around their restrictions to track users. In particular, two collaborating servers can often track users without using cookies at all by injecting identifying information into dynamic URLs.

7.2. 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 in outbound HTTP requests and the user agent MUST NOT process Set-Cookie headers in inbound HTTP responses.

Some user agents 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].

Some user agents provide users with the ability to approve individual writes to the cookie store. In many common usage scenarios, these controls generate a large number of prompts. However, some privacy-conscious users find these controls useful nonetheless.

7.3. 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 headers is transmitted in the clear.

1. All sensitive information conveyed in these headers is exposed to an eavesdropper.

2. A malicious intermediary could alter the headers as they travel in either direction, with unpredictable results.

3. A malicious client could alter the Cookie header 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 headers 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 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. 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 sent to https://site.example/ by impersonating a response from http://site.example/ and injecting a Set-Cookie header. 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. 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 "projects.example" has marked their session cookies as "SameSite", then this cross-site navigation won't send them along with the request. "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", and its absence would prompt a reauthentication step before executing any non-idempotent action. The former could drop the "SameSite" attribute entirely, or choose the "Lax" version of enforcement, in order to allow users access to data via top-level navigation.

8.8.3. Mashups and Widgets

The "SameSite" attribute is inappropriate for some important use-cases. In particular, note that content intended for embedding in a cross-site contexts (social networking widgets or commenting services, for instance) will not have access to same-site cookies. Cookies may be required for requests triggered in these cross-site contexts in order to provide seamless functionality that relies on a user's state.

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.

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).

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 5.5)

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 5.3)

9.3. Cookie Attribute Registry

The "Cookie Attribute Registry" defines the name space of attribute used to control cookies' behavior. The registry is maintained at https://www.iana.org/assignments/cookie-attribute-names [4].

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" will be updated 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	{{attribute-httponly} of this document
Max-Age	{{attribute-max-age} of this document
Path	{{attribute-path} of this document
SameSite	{{attribute-samesite} of this document
Secure	{{attribute-secure} of this document

10. References

10.1. Normative References

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

[HTML] Hickson, I., Pieters, S., van Kesteren, A., Jaegenstedt, P., and D. Denicola, "HTML", n.d., <https://html.spec.whatwg.org/>.

[RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, DOI 10.17487/RFC1034, November 1987, <https://www.rfc-editor.org/info/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/info/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/info/rfc2119>.

[RFC3490] Costello, A., "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

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/info/rfc4790>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/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/info/rfc5890>.

[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>.

[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>.

[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/info/rfc8126>.

[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", draft-ietf-httpbis-cookie-alone-01 (work in progress), September 2016.

[I-D.ietf-httpbis-cookie-prefixes]
West, M., "Cookie Prefixes", draft-ietf-httpbis-cookie-prefixes-00 (work in progress), February 2016.

[I-D.ietf-httpbis-cookie-same-site]
West, M. and M. Goodwin, "Same-Site Cookies", draft-ietf-httpbis-cookie-same-site-00 (work in progress), June 2016.

[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/>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/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/info/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/info/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/info/rfc3986>.

[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>.

[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/info/rfc5895>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/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/info/rfc7034>.

[UTS46] Davis, M. and M. Suignard, "Unicode IDNA Compatibility Processing", UNICODE Unicode Technical Standards # 46, June 2016, <http://unicode.org/reports/tr46/>.

10.3. URIs

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

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

[3] https://github.com/httpwg/http-extensions/labels/6265bis

[4] https://www.iana.org/assignments/cookie-attribute-names

[5] https://github.com/httpwg/http-extensions/issues/243

[6] https://github.com/httpwg/http-extensions/issues/246

[7] https://www.rfc-editor.org/errata_search.php?rfc=6265

[8] https://github.com/httpwg/http-extensions/issues/247

[9] https://github.com/httpwg/http-extensions/issues/201

[10] https://github.com/httpwg/http-extensions/issues/204

[11] https://github.com/httpwg/http-extensions/issues/222

[12] https://github.com/httpwg/http-extensions/issues/248

[13] https://github.com/httpwg/http-extensions/issues/295

[14] https://github.com/httpwg/http-extensions/issues/302

[15] https://github.com/httpwg/http-extensions/issues/389

[16] https://github.com/httpwg/http-extensions/issues/199

[17] https://github.com/httpwg/http-extensions/issues/788

[18] https://github.com/httpwg/http-extensions/issues/594

[19] https://github.com/httpwg/http-extensions/issues/159

[20] https://github.com/httpwg/http-extensions/issues/159

[21] https://github.com/httpwg/http-extensions/issues/901

[22] https://github.com/httpwg/http-extensions/pull/1035

[23] https://github.com/httpwg/http-extensions/pull/1038

[24] https://github.com/httpwg/http-extensions/pull/1040

[25] https://github.com/httpwg/http-extensions/pull/1047

[26] https://github.com/httpwg/http-extensions/issues/1059

[27] https://github.com/httpwg/http-extensions/issues/1158

[28] https://github.com/httpwg/http-extensions/pull/1060

[29] https://github.com/httpwg/http-extensions/issues/1074

[30] https://github.com/httpwg/http-extensions/issues/1119

[31] https://github.com/httpwg/http-extensions/pull/1143

[32] https://github.com/httpwg/http-extensions/issues/1159

Appendix A. Changes

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

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

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

- o Fixes to formatting caused by mistakes in the initial port to Markdown:

  * https://github.com/httpwg/http-extensions/issues/243 [5]

  * https://github.com/httpwg/http-extensions/issues/246 [6]

- o 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 [7]

- o Dropped "Cookie2" and "Set-Cookie2" from the IANA Considerations section: https://github.com/httpwg/http-extensions/issues/247 [8]

- o 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.

- o 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

- o Merged the recommendations from [I-D.ietf-httpbis-cookie-same-site], adding support for the "SameSite" attribute.

- o Closed a number of editorial bugs:

  * Clarified address bar behavior for SameSite cookies: https://github.com/httpwg/http-extensions/issues/201 [9]

* Added the word "Cookies" to the document's name: https://github.com/httpwg/http-extensions/issues/204 [10]

* Clarified that the "__Host-" prefix requires an explicit "Path" attribute: https://github.com/httpwg/http-extensions/issues/222 [11]

* 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 [12]

* Noted that double-quotes in cookie values are part of the value, and are not stripped: https://github.com/httpwg/http-extensions/issues/295 [13]

* Fixed the "site for cookies" algorithm to return something that makes sense: https://github.com/httpwg/http-extensions/issues/302 [14]

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

o Clarified handling of invalid SameSite values: https://github.com/httpwg/http-extensions/issues/389 [15]

o Reflect widespread implementation practice of including a cookie's "host-only-flag" when calculating its uniqueness: https://github.com/httpwg/http-extensions/issues/199 [16]

o Introduced an explicit "None" value for the SameSite attribute: https://github.com/httpwg/http-extensions/issues/788 [17]

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

o Allow "SameSite" cookies to be set for all top-level navigations. https://github.com/httpwg/http-extensions/issues/594 [18]

o Treat "Set-Cookie: token" as creating the cookie "("", "token")": https://github.com/httpwg/http-extensions/issues/159 [19]

o Reject cookies with neither name nor value (e.g. "Set-Cookie: =" and "Set-Cookie:": https://github.com/httpwg/http-extensions/issues/159 [20]

o Clarified behavior of multiple "SameSite" attributes in a cookie string: https://github.com/httpwg/http-extensions/issues/901 [21]

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

- Typos and editorial fixes: https://github.com/httpwg/http-extensions/pull/1035 [22], https://github.com/httpwg/http-extensions/pull/1038 [23], https://github.com/httpwg/http-extensions/pull/1040 [24], https://github.com/httpwg/http-extensions/pull/1047 [25].

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

- Editorial fixes: https://github.com/httpwg/http-extensions/issues/1059 [26], https://github.com/httpwg/http-extensions/issues/1158 [27].

- Created a registry for cookie attribute names: https://github.com/httpwg/http-extensions/pull/1060 [28].

- Tweaks to ABNF for "cookie-pair" and the "Cookie" header production: https://github.com/httpwg/http-extensions/issues/1074 [29], https://github.com/httpwg/http-extensions/issues/1119 [30].

- Fixed serialization for nameless/valueless cookies: https://github.com/httpwg/http-extensions/pull/1143 [31].

- Converted a normative reference to Mozilla's Public Suffix List [PSL] into an informative reference: https://github.com/httpwg/http-extensions/issues/1159 [32].

Acknowledgements

RFC 6265 was written by Adam Barth. This document is a minor update of RFC 6265, adding small 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

Mike West (editor)
Google, Inc

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

John Wilander (editor)
Apple, Inc

Email: wilander@apple.com

Network Working Group M. Nottingham
Internet-Draft May 11, 2017
Obsoletes: 3205 (if approved)
Intended status: Best Current Practice
Expires: November 12, 2017

On the use of HTTP as a Substrate
draft-nottingham-bcp56bis-00

Abstract

HTTP is often used as a substrate for other application protocols. This document specifies best practices for these protocols' use of HTTP.

Note to Readers

The issues list for this draft can be found at https://github.com/mnot/I-D/labels/bcp56bis .

The most recent (often, unpublished) draft is at https://mnot.github.io/I-D/bcp56bis/ .

Recent changes are listed at https://github.com/mnot/I-D/commits/gh-pages/bcp56bis .

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

HTTP [RFC7230] is often used as a substrate for other application protocols. This is done for a variety of reasons, including:

- familiarity by implementers, specifiers, administrators, developers and users,
- availability of a variety of client, server and proxy implementations,
- ease of use,
- ubiquity of Web browsers,
- reuse of existing mechanisms like authentication and encryption,
- presence of HTTP servers and clients in target deployments, and
- its ability to traverse firewalls.

The Internet community has a long tradition of protocol reuse, dating back to the use of Telnet [RFC0854] as a substrate for FTP [RFC0959] and SMTP [RFC2821]. However, layering new protocols over HTTP brings its own set of issues:

- Should an application using HTTP define a new URL scheme? Use new ports?
- Should it use standard HTTP methods and status codes, or define new ones?
- How can the maximum value be extracted from the use of HTTP?
- How does it coexist with other uses of HTTP - especially Web browsing?
- How can interoperability problems and "protocol dead ends" be avoided?

This document contains best current practices regarding the use of HTTP by applications other than Web browsing. Section 2 defines what applications it applies to; Section 3 surveys the properties of HTTP that are important to preserve, and Section 4 conveys best practices for those applications that do use HTTP.

It is written primarily to guide IETF efforts, but might be applicable in other situations. Note that the requirements herein do not necessarily apply to the development of generic HTTP extensions.

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. Is HTTP Being Used?

Different applications have different goals when using HTTP. In this document, we say an application is _using HTTP_ when any of the following conditions are true:

- The transport port in use is 80 or 443,
- The URL scheme "http" or "https" is used,
- The ALPN protocol ID [RFC7301] "http/1.1", "h2" or "h2c" is used, or
- The message formats described in [RFC7320] and/or [RFC7540] are used in conjunction with the IANA registries defined for HTTP.

When an application is using HTTP, all of the requirements of the HTTP protocol suite (including but not limited to [RFC7320], [RFC7321], [RFC7322], [RFC7233], [RFC7234], [RFC7325] and [RFC7540]) are in force.

An application might not be _using HTTP_ according to this definition, but still relying upon the HTTP specifications in some manner. For example, an application might wish to avoid re-specifying parts of the message format, but change others; or, it might want to use a different set of methods.

Such applications are referred to as _protocols based upon HTTP_ in this document. These have more freedom to modify protocol operation, but are also likely to lose at least a portion of the benefits outlined above, as most HTTP implementations won't be easily adaptable to these changes, and as the protocol diverges from HTTP, the benefit of mindshare will be lost.

Protocols that are based upon HTTP MUST NOT reuse HTTP's URL schemes, transport ports, ALPN protocol IDs or IANA registries; rather, they are encouraged to establish their own.

3. What’s Important About HTTP

There are many ways that HTTP applications are defined and deployed, and sometimes they are brought to the IETF for standardisation. In that process, what might be workable for deployment in a limited fashion isn’t appropriate for standardisation and the corresponding broader deployment.

This section examines the facets of the protocol that are important to preserve in these situations.

3.1. Generic Semantics

When writing an application’s specification, it’s often tempting to specify exactly how HTTP is to be implemented, supported and used.

However, this can easily lead to an unintended profile of HTTP’s behaviour. For example, it’s common to see specifications with language like this:

A ‘200 OK‘ response means that the widget has successfully been updated.

This sort of specification is bad practice, because it is adding new semantics to HTTP’s status codes and methods, respectively; a recipient - whether it’s an origin server, client library, intermediary or cache - now has to know these extra semantics to understand the message.

Some applications even require specific behaviours, such as:

A ‘POST‘ request MUST result in a ‘201 Created‘ response.

This forms an expectation in the client that the response will always be "201 Created", when in fact there are a number of reasons why the status code might differ in a real deployment. If the client does not anticipate this, the application’s deployment is brittle.

Much of the value of HTTP is in its _generic semantics_ - that is, the protocol elements defined by HTTP are potentially applicable to every resource, not specific to a particular context. Application-specific semantics are expressed in the payload; mostly, in the body, but also in header fields.

This allows a HTTP message to be examined by generic HTTP software (e.g., HTTP servers, intermediaries, client implementatiions), and its handling to be correctly determined. It also allows people to leverage their knowledge of HTTP semantics without special-casing them for a particular application.

Therefore, applications that use HTTP MUST NOT re-define, refine or overlay the semantics of defined protocol elements. Instead, they SHOULD focus their specifications on protocol elements that are specific to them; namely their HTTP resources.

See Section 4.2 for details.

3.2. Links

Another common practice is assuming that the HTTP server’s name space (or a portion thereof) is exclusively for the use of a single application. This effectively overlays special, application-specific semantics onto that space, precludes other applications from using it.

As explained in [RFC7320], such "squatting" on a part of the URL space by a standard usurps the server’s authority over its own resources, can cause deployment issues, and is therefore bad practice in standards.

Instead of statically defining URL paths, it is RECOMMENDED that applications using HTTP define links in payloads, to allow flexibility in deployment.

Using runtime links in this fashion has a number of other benefits. For example, navigating with a link allows a request to be routed to a different server without the overhead of a redirection, thereby supporting deployment across machines well. It becomes possible to "mix" different applications on the same server, and offers a natural path for extensibility, versioning and capability management.

3.3. Getting Value from HTTP

The simplest possible use of HTTP is to POST data to a single URL, thereby effectively tunnelling through the protocol.

This "RPC" style of communication does get some benefit from using HTTP - namely, message framing and the availability of implementations - but fails to realise many others:

- Caching for server scalability, latency and bandwidth reduction, and reliability;
- Authentication and access control;
- Automatic redirection;
- Partial content to selectively request part of a response;

- o Natural support for extensions and versioning through protocol extension; and
- o The ability to interact with the application easily using a Web browser.

Using such a high-level protocol to tunnel simple semantics has downsides too; because of its more advanced capabilities, breadth of deployment and age, HTTP's complexity can cause interoperability problems that could be avoided by using a simpler substrate (e.g., WebSockets [RFC6455], if browser support is necessary, or TCP [RFC0793] if not), or making the application be _based upon HTTP_, instead of using it (as defined in Section 2).

Applications that use HTTP are encouraged to accommodate the various features that the protocol offers, so that their users receive the maximum benefit from it. This document does not require specific features to be used, since the appropriate design tradeoffs are highly specific to a given situation. However, following the practices in Section 4 will help make them available.

4. Best Practices for Using HTTP

This section contains best practices regarding the use of HTTP by applications, including practices for specific HTTP protocol elements.

4.1. Specifying the Use of HTTP

When specifying the use of HTTP, an application SHOULD use [RFC7230] as the primary reference; it is not necessary to reference all of the specifications in the HTTP suite unless there are specific reasons to do so (e.g., a particular feature is called out).

Applications using HTTP MAY specify a minimum version to be supported (HTTP/1.1 is suggested), and MUST NOT specify a maximum version.

Likewise, applications need not specify what HTTP mechanisms - such as redirection, caching, authentication, proxy authentication, and so on - are to be supported. Full featured support for HTTP SHOULD be taken for granted in servers and clients, and the application's function SHOULD degrade gracefully if they are not (although this might be achieved by informing the user that their task cannot be completed).

For example, an application can specify that it uses HTTP like this:

Foo Application uses HTTP {{RFC7230}}. Implementations MUST support HTTP/1.1, and MAY support later versions. Support for common HTTP mechanisms such as redirection and caching are assumed.

4.2. Defining HTTP Resources

HTTP Applications SHOULD focus on defining the following application-specific protocol elements:

- o Media types [RFC6838], often based upon a format convention such as JSON [RFC7159],

- o HTTP header fields, as per Section 4.7, and

- o The behaviour of resources, as identified by link relations [RFC5988].

By composing these protocol elements, an application can define a set of resources, identified by link relations, that implement specified behaviours, including:

- o Retrieval of their state using GET, in one or more formats identified by media type;

- o Resource creation or update using POST or PUT, with an appropriately identified request body format;

- o Data processing using POST and identified request and response body format(s); and

- o Resource deletion using DELETE.

For example, an application might specify:

Resources linked to with the "example-widget" link relation type are Widgets. The state of a Widget can be fetched in the "application/example-widget+json" format, and can be updated by PUT to the same link. Widget resources can be deleted.

The "Example-Count" response header field on Widget representations indicates how many Widgets are held by the sender.

The "application/example-widget+json" format is a JSON {{RFC7159}} format representing the state of a Widget. It contains links to related information in the link indicated by the Link header field value with the "example-other-info" link relation type.

4.3. HTTP URLs

In HTTP, URLs are opaque identifiers under the control of the server. As outlined in [RFC7320], standards cannot usurp this space, since it might conflict with existing resources, and constrain implementation and deployment.

In other words, applications that use HTTP MUST NOT associate application semantics with specific URL paths. For example, specifying that a "GET to the URL /foo retrieves a bar document" is bad practice. Likewise, specifying "The widget API is at the path /bar" violates [RFC7320].

Instead, applications that use HTTP are encouraged to use typed links [RFC5988] to convey the URIs that are in use, as well as the semantics of the resources that they identify. See Section 4.2 for details.

4.3.1. Initial URL Discovery

Generally, a client with begin interacting with a given application server by requesting an initial document that contains information about that particular deployment, potentially including links to other relevant resources.

Applications that use HTTP SHOULD allow an arbitrary URL to be used as that entry point. For example, rather than specifying "the initial document is at "/foo/v1", they should allow a deployment to use any URL as the entry point for the application.

In cases where doing so is impractical (e.g., it is not possible to convey a whole URL, but only a hostname) applications that use HTTP MAY define a well-known URL [RFC5785] as an entry point.

4.3.2. URL Schemes

Applications that use HTTP MUST allow use of the "https" URL scheme, and SHOULD NOT allow use of the "http" URL scheme, unless interoperability considerations with existing deployments require it. They MUST NOT use other URL schemes.

"https" is preferred to mitigate pervasive monitoring attacks [RFC7258].

Using other schemes to denote an application using HTTP makes it more difficult to use with existing implementations (e.g., Web browsers), and is likely to fail to meet the requirements of [RFC7595].

If it is necessary to advertise the application in use, this SHOULD be done in message payloads, not the URL scheme.

4.3.3. Transport Ports

Applications that use HTTP SHOULD use the default port for the URL scheme in use. If it is felt that networks might need to distinguish the application’s traffic for operational reasons, it MAY register a separate port, but be aware that this has privacy implications for that protocol’s users. The impact of doing so MUST be documented in Security Considerations.

4.4. Authentication and Application State

Applications that use HTTP MAY use stateful cookies [RFC6265] to identify a client and/or store client-specific data to contextualise requests.

If it is only necessary to identify clients, applications that use HTTP MAY use HTTP authentication [RFC7235]; if the Basic authentication scheme [RFC7617] is used, it MUST NOT be used with the ’http’ URL scheme.

In either case, it is important to carefully specify the scoping and use of these mechanisms; if they expose sensitive data or capabilities (e.g., by acting as an ambiant authority), exploits are possible. Mitigations include using a request-specific token to assure the intent of the client.

4.5. HTTP Methods

Applications that use HTTP MUST confine themselves to using registered HTTP methods such as GET, POST, PUT, DELETE, and PATCH.

New HTTP methods are rare; they are required to be registered with IETF Review (see [RFC7232]), and are also required to be _generic_. That means that they need to be potentially applicable to all resources, not just those of one application.

While historically some applications (e.g., [RFC6352] and [RFC4791]) have defined non-generic methods, [RFC7231] now forbids this.

When it is believed that a new method is required, authors are encouraged to engage with the HTTP community early, and document their proposal as a separate HTTP extension, rather than as part of an application’s specification.

4.6. HTTP Status Codes

Applications that use HTTP MUST only use registered HTTP status codes.

As with methods, new HTTP status codes are rare, and required (by [RFC7231]) to be registered with IETF review. Similarly, HTTP status codes are generic; they are required (by [RFC7231]) to be potentially applicable to all resources, not just to those of one application.

When it is believed that a new status code is required, authors are encouraged to engage with the HTTP community early, and document their proposal as a separate HTTP extension, rather than as part of an application's specification.

Status codes' primary function is to convey HTTP semantics for the benefit of generic HTTP software, not application-specific semantics. Therefore, applications MUST NOT specify additional semantics or refine existing semantics for status codes.

In particular, specifying that a particular status code has a specific meaning in the context of an application is harmful, as these are not generic semantics, since the consumer needs to be in the context of the application to understand them.

Furthermore, applications using HTTP MUST NOT re-specify the semantics of HTTP status codes, even if it is only by copying their definition. They MUST NOT require specific status phrases to be used; the status phrase has no function in HTTP, and is not guaranteed to be preserved by implementations.

Typically, applications using HTTP will convey application-specific information in the message body and/or HTTP header fields, not the status code.

Specifications sometimes also create a "laundry list" of potential status codes, in an effort to be helpful. The problem with doing so is that such a list is never complete; for example, if a network proxy is interposed, the client might encounter a "407 Proxy Authentication Required" response; or, if the server is rate limiting the client, it might receive a "429 Too Many Requests" response.

Since the list of HTTP status codes can be added to, it's safer to refer to it directly, and point out that clients SHOULD be able to handle all applicable protocol elements gracefully (i.e., falling back to the generic "n00" semantics of a given status code; e.g., "499" can be safely handled as "400" by clients that don't recognise it).

4.7. HTTP Header Fields

Applications that use HTTP MAY define new HTTP header fields, following the advice in [RFC7321], Section 8.3.1.

Typically, using HTTP header fields is appropriate in a few different situations:

- o Their content is useful to intermediaries (who often wish to avoid parsing the body), and/or
- o Their content is useful to generic HTTP software (e.g., clients, servers), and/or
- o It is not possible to include their content in the message body (usually because a format does not allow it).

If none of these motivations apply, using a header field is NOT RECOMMENDED.

New header fields MUST be registered, as per [RFC7231] and [RFC3864].

It is RECOMMENDED that header field names be short (even when HTTP/2 header compression is in effect, there is an overhead) but appropriately specific. In particular, if a header field is specific to an application, an identifier for that application SHOULD form a prefix to the header field name, separated by a "-".

The semantics of existing HTTP header fields MUST NOT be re-defined without updating their registration or defining an extension to them (if allowed). For example, an application using HTTP cannot specify that the "Location" header has a special meaning in a certain context.

See Section 4.4 for requirements regarding header fields that carry application state (e.g,. Cookie).

5. IANA Considerations

This document has no requirements for IANA.

6. Security Considerations

Section 4.4 discusses the impact of using stateful mechanisms in the protocol as ambiant authority, and suggests a mitigation.

Section 4.3.2 requires support for ’https’ URLs, and discourages the use of ’http’ URLs, to mitigate pervasive monitoring attacks.

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>.

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

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

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

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

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

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

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <http://www.rfc-editor.org/info/rfc7301>.

[RFC7320] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <http://www.rfc-editor.org/info/rfc7320>.

[RFC7321] McGrew, D. and P. Hoffman, "Cryptographic Algorithm Implementation Requirements and Usage Guidance for Encapsulating Security Payload (ESP) and Authentication Header (AH)", RFC 7321, DOI 10.17487/RFC7321, August 2014, <http://www.rfc-editor.org/info/rfc7321>.

[RFC7322] Flanagan, H. and S. Ginoza, "RFC Style Guide", RFC 7322, DOI 10.17487/RFC7322, September 2014, <http://www.rfc-editor.org/info/rfc7322>.

[RFC7325] Villamizar, C., Ed., Kompella, K., Amante, S., Malis, A., and C. Pignataro, "MPLS Forwarding Compliance and Performance Requirements", RFC 7325, DOI 10.17487/RFC7325, August 2014, <http://www.rfc-editor.org/info/rfc7325>.

[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>.

[RFC7595] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <http://www.rfc-editor.org/info/rfc7595>.

7.2. Informative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

[RFC0854] Postel, J. and J. Reynolds, "Telnet Protocol Specification", STD 8, RFC 854, DOI 10.17487/RFC0854, May 1983, <http://www.rfc-editor.org/info/rfc854>.

[RFC0959] Postel, J. and J. Reynolds, "File Transfer Protocol", STD 9, RFC 959, DOI 10.17487/RFC0959, October 1985, <http://www.rfc-editor.org/info/rfc959>.

[RFC2821] Klensin, J., Ed., "Simple Mail Transfer Protocol", RFC 2821, DOI 10.17487/RFC2821, April 2001, <http://www.rfc-editor.org/info/rfc2821>.

[RFC4791] Daboo, C., Desruisseaux, B., and L. Dusseault, "Calendaring Extensions to WebDAV (CalDAV)", RFC 4791, DOI 10.17487/RFC4791, March 2007, <http://www.rfc-editor.org/info/rfc4791>.

[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>.

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

[RFC6352] Daboo, C., "CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV)", RFC 6352, DOI 10.17487/RFC6352, August 2011, <http://www.rfc-editor.org/info/rfc6352>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <http://www.rfc-editor.org/info/rfc6455>.

[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>.

[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>.

[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>.

[RFC7617] Reschke, J., "The ’Basic’ HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <http://www.rfc-editor.org/info/rfc7617>.

Author's Address

Mark Nottingham

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

HTTP Working Group   B. Schwartz
Internet-Draft   Google
Intended status: Standards Track   M. Bishop
Expires: October 25, 2018   Akamai Technologies
                                                          April 23, 2018

Finding HTTP Alternative Services via the Domain Name Service
draft-schwartz-httpbis-dns-alt-svc-02

Abstract

The HTTP Alternative Services (Alt-Svc) mechanism allows an HTTP origin to be served from multiple network endpoints, and over multiple protocols. However, the client must first contact the origin server, in order to learn of the alternative services. This draft proposes a straightforward mapping of Alt-Svc into DNS, allowing clients to learn of these services before their first contact with the origin. This arrangement offers potential benefits to both performance and privacy.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

The HTTP Alternative Services standard [AltSvc] defines

- o an extensible data model for describing alternative network endpoints that are authoritative for an origin
- o the "Alt-Svc Field Value", a text format for representing this information
- o standards for sending information in this format from a server to a client over HTTP/1.1 and HTTP/2.

Together, these components provide a toolkit that has proven useful and effective for informing a client of alternative services for an origin. However, making use of an alternative service requires contacting the origin server first. This creates an obvious performance cost: users wait for a full HTTP connection initiation (multiple roundtrips) before learning of an alternative service that is preferred by the origin. The first connection also publicly reveals the user's intended destination to all entities along the network path.

This draft proposes a straightforward mechanism to distribute the Alt-Svc Field Value, in its standard text format, through the DNS. If a client receives this information during DNS resolution, it can skip the initial connection and proceed directly to an alternative service.

1.1. Terminology

For consistency with [AltSvc], we adopt the following definitions

- An "origin" is an information source as in [RFC6454].
- The "origin server" is the server that the client would reach when accessing the origin in the absence of Alt-Svc.
- An "alternative service" is a different server that can serve the origin.

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 ALTSVC record type

The ALTSVC DNS resource record (RR) type (RRTYPE ???) is used to associate an Alternative Service Field Value with an origin. Abstractly, the origin consists of a scheme (typically "https"), a host name, and a port (typically "443").

In the case of the ALTSVC RR, the origin is represented by prefixing the port and scheme with "_", then concatenating them with the host, resulting in a domain name like "_443._https.www.example.com.".

The RDATA portion of an ALTSVC resource record contains an Alt-Svc Field Value, exactly as defined in Section 4 of [AltSvc].

For example, if the operator of https://www.example.com intends to include an HTTP response header like

```
Alt-Svc: h2=":8000"; ma=60
```

They would also publish an ALTSVC DNS record like

```
_443._https.www.example.com. 60S IN ALTSVC "h2=\":8000\""
```

This data type can be represented as an Unknown RR as described in [RFC3597]:

```
_443._https.www.example.com. 60S IN TYPE??? \# 10
68323D223A3830303022
```

This construction is intended to be extensible in two ways. First, any extensions that are made to the Alt-Svc format for transmission over HTTPS are also applicable here, unless expressly mentioned otherwise. Second, including the scheme in the DNS name allows for ALTSVC to serve schemes other than HTTPS, such as HTTP with Opportunistic Security [RFC8164] and any future schemes for which Alt-Svc may be defined.

2.1. Comparison with alternatives

The ALTSVC record type closely resembles some existing record types.

2.1.1. Differences from the SRV RRTYPE

An SRV record can perform a similar function to the ALTSVC record, informing a client to look in a different location for a service. However, there are several differences:

- o SRV records are typically mandatory, whereas clients will always continue to function correctly without making use of Alt-Svc.

- o SRV records cannot instruct the client to switch or upgrade protocols, whereas Alt-Svc can signal such an upgrade (e.g. to HTTP/2).

- o SRV records are not extensible, whereas Alt-Svc can be extended with new parameters. For example, this is what allows the privacy improvements related to SNI selection in [AltSvcSNI].

- o Using SRV records would not allow a client to skip processing of the Alt-Svc information in a subsequent connection, so it does not confer a performance advantage.

2.1.2. Differences from the TXT RRTYPE

The ALTSVC record uses an identical format to a TXT record, and could be implemented as such. However, we define a new record type for clarity, and to respect the use of TXT for human-readable notes as recommended in [RFC5507].

3. Differences from Alt-Svc as transmitted over HTTP

Publishing an ALTSVC record in DNS is intended to be equivalent to transmitting this field value over HTTP, and receiving an ALTSVC record is intended to be equivalent to receiving this field value over HTTP. However, there are some small differences in the intended client and server behavior.

3.1. Omitting Max Age

When publishing an ALTSVC record in DNS, server operators MUST omit the "ma" parameter, which encodes the "max age" (i.e. expiration time) of an Alt-Svc Field Value. Instead, server operators SHOULD encode the expiration time in the DNS TTL, and MUST NOT set a TTL longer than the intended "max age".

Server operators MAY publish multiple ALTSVC records as an RRSET, with semantics equivalent to other mechanisms of providing multiple Alt-Svc values to the client. When publishing an RRSET with multiple ALTSVC records, the server operator MUST set the overall TTL to the minimum of the "max age" values (following Section 5.2 of [RFC2181]).

When receiving an ALTSVC record, clients MAY synthesize a new "ma" parameter from the DNS TTL, in order to interoperate with Alt-Svc processing subsystems.

3.2. Interaction with other standards

The purpose of this standard is to reduce connection latency and improve user privacy. Server operators implementing this standard SHOULD also implement TLS 1.3 [I-D.ietf-tls-tls13] and OCSP Stapling [RFC6066], both of which confer substantial performance and privacy benefits when used in combination with ALTSVC records.

To realize the greatest privacy benefits, this proposal is intended for use with a privacy-preserving DNS transport (like DNS over TLS [RFC7858] or DNS over HTTPS [DOH]), and with the "SNI" Alt-Svc Parameter [AltSvcSNI]. However, performance improvements, and some modest privacy improvements, are possible without the use of those standards.

3.3. Granularity and lifetime control

Sending Alt-Svc over HTTP allows the server to tailor the Alt-Svc Field Value specifically to the client. When using an ALTSVC DNS record, groups of clients will necessarily receive the same Alt-Svc Field Value. Therefore, this standard is not suitable for servers that require single-client granularity in Alt-Svc. Server operators that want to serve different Alt-Svc Field Values to different geographic or network regions SHOULD configure their authoritative DNS server to respect the EDNS0 Client Subnet extension [RFC7871].

Some DNS caching systems incorrectly extend the lifetime of DNS records beyond the stated TTL. Server operators MUST NOT rely on ALTSVC records expiring on time, and MAY shorten the TTL to compensate.

4. Client behaviors

4.1. Cache interaction

If the client has an Alt-Svc cache, and a usable Alt-Svc value is present in that cache, then the client SHOULD NOT issue an ALTSVC DNS query. Instead, the client SHOULD proceed with alternative service connection as usual.

If the client has a cached Alt-Svc entry that is expiring, the client MAY perform an ALTSVC query to refresh the entry.

4.2. Optimizing for performance

Clients that are optimizing for performance (i.e. minimum connection setup time) SHOULD implement the following connection sequence:

1. Issue address (AAAA and/or A) queries, immediately followed by the ALTSVC query.

2. If an ALTSVC response is received first, proceed with alternative service connection and ignore the address responses if they are no longer relevant.

3. Otherwise, initiate connection to the origin server.

4. As soon as an Alt-Svc field value is received, through the DNS or over HTTP, proceed with alternative service connection. Do not abort this connection if an Alt-Svc field value is received from the other source later.

If the ALTSVC and address queries return approximately simultaneously, this process typically saves three roundtrips on a fresh connection that uses Alt-Svc: one each for TCP, TLS 1.3, and HTTP. (On subsequent connections, the Alt-Svc information is expected to be cached, so this procedure does not apply.)

If a client can cache Alt-Svc entries that were received over both HTTP and DNS, the client MAY prefer entries that were received over

HTTP. These records may be more narrowly targeted for the specific client.

As an additional optimization, when choosing among multiple Alt-Svc values, clients MAY prefer those that will not require an address query, either because the corresponding address record is already in cache or because the host is an IP address.

Note that this procedure does not rely on recursive resolvers handling the ALTSVC record type correctly. If ALTSVC queries receive spurious NXDOMAIN responses, or even no response at all, connections will proceed as usual without any delay.

4.3. Optimizing for privacy

Clients that are optimizing for privacy SHOULD implement [AltSvcSNI] and DNS over a secure transport (e.g. [RFC7858] or [DOH]). Use of a secure transport is important not only for privacy protection, but also to ensure that queries for the new ALTSVC RRTYPE are handled correctly. Additionally, these clients SHOULD implement the following connection sequence:

1. Issue the ALTSVC DNS query first, immediately followed by the address queries.

2. Wait for the ALTSVC record response.

3. If the response is nonempty, proceed with alternative service connection and ignore the address query responses.

4. Otherwise, wait for the address queries and connect as usual.

Note that this process is also expected to be faster than Alt-Svc over HTTP in the case of HTTP Opportunistic Upgrade Probing (Section 2 of [RFC8164]).

5. Security Considerations

Alt-Svc Field Values are intended for distribution over untrusted channels, and clients are REQUIRED to verify that the alternative service is authoritative for the origin (Section 2.1 of [AltSvc]). Therefore, DNSSEC signing and validation are OPTIONAL for publishing and using ALTSVC records.

6. IANA Considerations

This draft requires assignment of a new DNS RRTYPE value.

7. References

7.1. Normative References

[AltSvc] 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>.

[AltSvcSNI] Bishop, M., "The "SNI" Alt-Svc Parameter", draft-bishop-httpbis-sni-altsvc-01 (work in progress), January 2018.

[DOH] Hoffman, P. and P. McManus, "DNS Queries over HTTPS", draft-ietf-doh-dns-over-https-07 (work in progress), April 2018.

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

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC2181] Elz, R. and R. Bush, "Clarifications to the DNS Specification", RFC 2181, DOI 10.17487/RFC2181, July 1997, <https://www.rfc-editor.org/info/rfc2181>.

[RFC3597] Gustafsson, A., "Handling of Unknown DNS Resource Record (RR) Types", RFC 3597, DOI 10.17487/RFC3597, September 2003, <https://www.rfc-editor.org/info/rfc3597>.

[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>.

[RFC7858] Hu, Z., Zhu, L., Heidemann, J., Mankin, A., Wessels, D., and P. Hoffman, "Specification for DNS over Transport Layer Security (TLS)", RFC 7858, DOI 10.17487/RFC7858, May 2016, <https://www.rfc-editor.org/info/rfc7858>.

[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

[RFC5507] IAB, Faltstrom, P., Ed., Austein, R., Ed., and P. Koch, Ed., "Design Choices When Expanding the DNS", RFC 5507, DOI 10.17487/RFC5507, April 2009, <https://www.rfc-editor.org/info/rfc5507>.

[RFC7871] Contavalli, C., van der Gaast, W., Lawrence, D., and W. Kumari, "Client Subnet in DNS Queries", RFC 7871, DOI 10.17487/RFC7871, May 2016, <https://www.rfc-editor.org/info/rfc7871>.

[RFC8164] Nottingham, M. and M. Thomson, "Opportunistic Security for HTTP/2", RFC 8164, DOI 10.17487/RFC8164, May 2017, <https://www.rfc-editor.org/info/rfc8164>.

Authors' Addresses

Ben Schwartz
Google

Email: bemasc@google.com

Mike Bishop
Akamai Technologies

Email: mbishop@evequefou.be

Network Working Group   J. Yasskin
Internet-Draft   Google
Intended status: Standards Track   27 July 2020
Expires: 28 January 2021

Signed HTTP Exchanges
draft-yasskin-http-origin-signed-responses-09

Abstract

This document specifies how a server can send an HTTP exchange--a request URL, content negotiation information, and a response--with signatures that vouch for that exchange's authenticity. These signatures can be verified against an origin's certificate to establish that the exchange is authoritative for an origin even if it was transferred over a connection that isn't. The signatures can also be used in other ways described in the appendices.

These signatures contain countermeasures against downgrade and protocol-confusion attacks.

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/ (https://lists.w3.org/Archives/Public/ietf-http-wg/).

The source code and issues list for this draft can be found in https://github.com/WICG/webpackage (https://github.com/WICG/webpackage).

This Internet-Draft will expire on 28 January 2021.

Copyright Notice

Table of Contents

1. Introduction

Signed HTTP exchanges provide a way to prove the authenticity of a resource in cases where the transport layer isn't sufficient. This can be used in several ways:

* When signed by a certificate ([RFC5280]) that's trusted for an origin, an exchange can be treated as authoritative for that origin, even if it was transferred over a connection that isn't authoritative (Section 9.1 of [RFC7230]) for that origin. See Appendix A.1 and Appendix A.2.

* A top-level resource can use a public key to identify an expected publisher for particular subresources, a system known as Subresource Integrity ([SRI]). An exchange's signature provides the matching proof of authorship. See Appendix A.3.

* A signature can vouch for the exchange in some way, for example that it appears in a transparency log or that static analysis indicates that it omits certain attacks. See Appendix A.4 and Appendix A.5.

Subsequent work toward the use cases in [I-D.yasskin-wpack-use-cases] will provide a way to group signed exchanges into bundles that can be transmitted and stored together, but single signed exchanges are useful enough to standardize on their own.

2. Terminology

Absolute URL A string for which the URL parser (https://url.spec.whatwg.org/#concept-url-parser) ([URL]), when run without a base URL, returns a URL rather than a failure, and for which that URL has a null fragment. This is similar to the absolute-URL string (https://url.spec.whatwg.org/#absolute-url-string) concept defined by ([URL]) but might not include exactly the same strings.

Author The entity that wrote the content in a particular resource. This specification deals with publishers rather than authors.

Publisher The entity that controls the server for a particular origin [RFC6454]. The publisher can get a CA to issue certificates for their private keys and can run a TLS server for their origin.

Exchange (noun) An HTTP request URL, content negotiation information, and an HTTP response. This can be encoded into a request message from a client with its matching response from a server, into the request in a PUSH_PROMISE with its matching response stream, or into the dedicated format in Section 5.3, which uses [I-D.ietf-httpbis-variants] to encode the content negotiation information. This is not quite the same meaning as defined by Section 8 of [RFC7540], which assumes the content negotiation information is embedded into HTTP request headers.

Intermediate An entity that fetches signed HTTP exchanges from a publisher or another intermediate and forwards them to another intermediate or a client.

Client An entity that uses a signed HTTP exchange and needs to be able to prove that the publisher vouched for it as coming from its claimed origin.

Unix time Defined by [POSIX] section 4.16 (http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_16).

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.

3. Signing an exchange

In the response of an HTTP exchange the server MAY include a "Signature" header field (Section 3.1) holding a list of one or more parameterised signatures that vouch for the content of the exchange. Exactly which content the signature vouches for can depend on how the exchange is transferred (Section 5).

The client categorizes each signature as "valid" or "invalid" by validating that signature with its certificate or public key and other metadata against the exchange’s URL, response headers, and content (Section 3.5). This validity then informs higher-level protocols.

Each signature is parameterised with information to let a client fetch assurance that a signed exchange is still valid, in the face of revoked certificates and newly-discovered vulnerabilities. This assurance can be bundled back into the signed exchange and forwarded to another client, which won’t have to re-fetch this validity information for some period of time.

3.1. The Signature Header

The "Signature" header field conveys a list of signatures for an exchange, each one accompanied by information about how to determine the authority of and refresh that signature. Each signature directly signs the exchange’s URL and response headers and identifies one of those headers that enforces the integrity of the exchange’s payload.

The "Signature" header is a Structured Header as defined by [I-D.ietf-httpbis-header-structure]. Its value MUST be a parameterised list (Section 3.4 of [I-D.ietf-httpbis-header-structure]). Its ABNF is:

```
Signature = sh-param-list
```

Each parameterised identifier in the list MUST have parameters named "sig", "integrity", "validity-url", "date", and "expires". Each parameterised identifier MUST also have either "cert-url" and "cert-sha256" parameters or an "ed25519key" parameter. This specification gives no meaning to the identifier itself, which can be used as a human-readable identifier for the signature (however, this is likely to change soon; see Section 3.1.2, Paragraph 1). The present parameters MUST have the following values:

"sig" Byte sequence (Section 3.10 of [I-D.ietf-httpbis-header-structure]) holding the signature of most of these parameters and the exchange’s URL and response headers.

"integrity" A string (Section 3.8 of

[I-D.ietf-httpbis-header-structure]) containing a "/"-separated sequence of names starting with the lowercase name of the response header field that guards the response payload's integrity. The meaning of subsequent names depends on the response header field, but for the "digest" header field, the single following name is the name of the digest algorithm that guards the payload's integrity.

"cert-url" A string (Section 3.8 of [I-D.ietf-httpbis-header-structure]) containing an absolute URL (Section 2) with a scheme of "https" or "data".

"cert-sha256" Byte sequence (Section 3.10 of [I-D.ietf-httpbis-header-structure]) holding the SHA-256 hash of the first certificate found at "cert-url".

"ed25519key" Byte sequence (Section 3.10 of [I-D.ietf-httpbis-header-structure]) holding an Ed25519 public key ([RFC8032]).

"validity-url" A string (Section 3.8 of [I-D.ietf-httpbis-header-structure]) containing an absolute URL (Section 2) with a scheme of "https".

"date" and "expires" An integer (Section 3.6 of [I-D.ietf-httpbis-header-structure]) representing a Unix time.

The "cert-url" parameter is _not_ signed, so intermediates can update it with a pointer to a cached version.

3.1.1. Examples

The following header is included in the response for an exchange with effective request URI "https://example.com/resource.html". Newlines are added for readability.

```
Signature:
 sig1;
  sig=*MEUCIQDXlI2gN3RNBlgFiuRNFpZXcDIaUpX6HIEwcZEc0cZYLAIga9DsVOMM+g5YpwEBdGW3sS
+bvnmAJJiSMwhuBdqp5UY=*;
  integrity="digest/mi-sha256";
  validity-url="https://example.com/resource.validity.1511128380";
  cert-url="https://example.com/oldcerts";
  cert-sha256=*W7uB969dFW3Mb5ZefPS9Tq5ZbH5iSmOILpjv2qEArmI=*;
  date=1511128380; expires=1511733180,
 sig2;
  sig=*MEQCIGjZRqTRf9iKNkGFyzRMTFgwf/BrY2ZNIP/dykhUV0aYAiBTXg+8wujoT4n/W+cNgb7pGq
QvIUGYZ8u8HZJ5YH26Qg==*;
  integrity="digest/mi-sha256";
  validity-url="https://example.com/resource.validity.1511128380";
  cert-url="https://example.com/newcerts";
  cert-sha256=*J/lEm9kNRODdCmINbvitpvdYKNQ+YgBj99DlYp4fEXw=*;
  date=1511128380; expires=1511733180,
 srisig;
  sig=*lGZVaJJM5f2oGczFlLmBdKTDL+QADza4BgeO494ggACYJOvrof6uh5OJCcwKrk7DK+LBch0jss
DYPp5CLc1SDA==*;
  integrity="digest/mi-sha256";
  validity-url="https://example.com/resource.validity.1511128380";
  ed25519key=*zsSevyFsxyZHiUluVBDd4eypdRLTqyWRVOJuuKUz+A8=*
  date=1511128380; expires=1511733180,
 thirdpartysig;
  sig=*MEYCIQCNxJzn6Rh2fNxsobktir8TkiaJYQFhWTuWI1i4PewQaQIhAMs2TVjc4rTshDtXbgQEOw
gj2mRXALhfXPztXgPupii+*;
  integrity="digest/mi-sha256";
  validity-url="https://thirdparty.example.com/resource.validity.1511161860";
  cert-url="https://thirdparty.example.com/certs";
  cert-sha256=*UeOwUPkvxlGRTyvHcsMUN0A2oNsZbU8EUvg8A9ZAnNc=*;
  date=1511133060; expires=1511478660,
```

There are 4 signatures: 2 from different secp256r1 certificates within "https://example.com/", one using a raw ed25519 public key that’s also controlled by "example.com", and a fourth using a secp256r1 certificate owned by "thirdparty.example.com".

All 4 signatures rely on the "Digest" response header with the mi-sha256 digest algorithm to guard the integrity of the response payload.

The signatures include a "validity-url" that includes the first time the resource was seen. This allows multiple versions of a resource at the same URL to be updated with new signatures, which allows clients to avoid transferring extra data while the old versions don’t have known security bugs.

The certificates at "https://example.com/oldcerts" and "https://example.com/newcerts" have "subjectAltName"s of "example.com", meaning that if they and their signatures validate, the exchange can be trusted as having an origin of "https://example.com/". The publisher might be using two certificates because their readers have disjoint sets of roots in their trust stores.

The publisher signed with all three certificates at the same time, so they share a validity range: 7 days starting at 2017-11-19 21:53 UTC.

The publisher then requested an additional signature from "thirdparty.example.com", which did some validation or processing and then signed the resource at 2017-11-19 23:11 UTC. "thirdparty.example.com" only grants 4-day signatures, so clients will need to re-validate more often.

3.1.2. Open Questions

The next revision of [I-D.ietf-httpbis-header-structure] will provide a way to parameterise byte sequences, at which point the signature itself is likely to become the main list item.

Should the cert-url and validity-url be lists so that intermediates can offer a cache without losing the original URLs? Putting lists in dictionary fields is more complex than [I-D.ietf-httpbis-header-structure] allows, so they're single items for now.

3.2. CBOR representation of exchange response headers

To sign an exchange's response headers, they need to be serialized into a byte string. Since intermediaries and distributors (Appendix A.2) might rearrange, add, or just reserialize headers, we can't use the literal bytes of the headers as this serialization. Instead, this section defines a CBOR representation that can be embedded into other CBOR, canonically serialized (Section 3.4), and then signed.

The CBOR representation of a set of response metadata and headers is the CBOR ([RFC7049]) map with the following mappings:

* The byte string ':status' to the byte string containing the response's 3-digit status code, and

* For each response header field, the header field's lowercase name as a byte string to the header field's value as a byte string.

3.2.1. Example

Given the HTTP exchange:

```
GET / HTTP/1.1
Host: example.com
Accept: */*

HTTP/1.1 200
Content-Type: text/html
Digest: mi-sha256=dcRDgR2GM35DluAV13PzgnG6+pvQwPywfFvAu1UeFrs=
Signed-Headers: "content-type", "digest"

<!doctype html>
<html>
...
```

The cbor representation consists of the following item, represented using the extended diagnostic notation from [CDDL] appendix G:

```
{
  'digest': 'mi-sha256=dcRDgR2GM35DluAV13PzgnG6+pvQwPywfFvAu1UeFrs=',
  ':status': '200',
  'content-type': 'text/html'
}
```

3.3. Loading a certificate chain

The resource at a signature's "cert-url" MUST have the "application/cert-chain+cbor" content type, MUST be canonically-encoded CBOR (Section 3.4), and MUST match the following CDDL:

```
cert-chain = [
  "", ; U+1F4DC U+26D3
  + augmented-certificate
]
augmented-certificate = {
  cert: bytes,
  ? ocsp: bytes,
  ? sct: bytes,
  * tstr => any,
}
```

The first map (second item) in the CBOR array is treated as the end-entity certificate, and the client will attempt to build a path ([RFC5280]) to it from a trusted root using the other certificates in the chain.

1. Each "cert" value MUST be a DER-encoded X.509v3 certificate ([RFC5280]). Other key/value pairs in the same array item define properties of this certificate.

2. The first certificate’s "ocsp" value MUST be a complete, DER-encoded OCSP response for that certificate (using the ASN.1 type "OCSPResponse" defined in [RFC6960]). Subsequent certificates MUST NOT have an "ocsp" value.

3. Each certificate’s "sct" value if any MUST be a "SignedCertificateTimestampList" for that certificate as defined by Section 3.3 of [RFC6962].

Loading a "cert-url" takes a "forceFetch" flag. The client MUST:

1. Let "raw-chain" be the result of fetching ([FETCH]) "cert-url". If "forceFetch" is _not_ set, the fetch can be fulfilled from a cache using normal HTTP semantics [RFC7234]. If this fetch fails, return "invalid".

2. Let "certificate-chain" be the array of certificates and properties produced by parsing "raw-chain" using the CDDL above. If any of the requirements above aren’t satisfied, return "invalid". Note that this validation requirement might be impractical to completely achieve due to certificate validation implementations that don’t enforce DER encoding or other standard constraints.

3. Return "certificate-chain".

3.4. Canonical CBOR serialization

Within this specification, the canonical serialization of a CBOR item uses the following rules derived from Section 3.9 of [RFC7049] with erratum 4964 applied:

* Integers and the lengths of arrays, maps, and strings MUST use the smallest possible encoding.

* Items MUST NOT be encoded with indefinite length.

* The keys in every map MUST be sorted in the bytewise lexicographic order of their canonical encodings. For example, the following keys are correctly sorted:

    1. 10, encoded as 0A.

    2. 100, encoded as 18 64.

    3. -1, encoded as 20.

    4. "z", encoded as 61 7A.

5. "aa", encoded as 62 61 61.

6. [100], encoded as 81 18 64.

7. [-1], encoded as 81 20.

8. false, encoded as F4.

Note: this specification does not use floating point, tags, or other more complex data types, so it doesn't need rules to canonicalize those.

3.5. Signature validity

The client MUST parse the "Signature" header field as the parameterised list (Section 4.2.5 of [I-D.ietf-httpbis-header-structure]) described in Section 3.1. If an error is thrown during this parsing or any of the requirements described there aren't satisfied, the exchange has no valid signatures. Otherwise, each member of this list represents a signature with parameters.

The client MUST use the following algorithm to determine whether each signature with parameters is invalid or potentially-valid for an exchange's

* "requestUrl", a byte sequence that can be parsed into the exchange's effective request URI (Section 5.5 of [RFC7230]),

* "responseHeaders", a byte sequence holding the canonical serialization (Section 3.4) of the CBOR representation (Section 3.2) of the exchange's response metadata and headers, and

* "payload", a stream of bytes constituting the exchange's payload body (Section 3.3 of [RFC7230]). Note that the payload body is the message body with any transfer encodings removed.

Potentially-valid results include:

* The signed headers of the exchange so that higher-level protocols can avoid relying on unsigned headers, and

* Either a certificate chain or a public key so that a higher-level protocol can determine whether it's actually valid.

This algorithm accepts a "forceFetch" flag that avoids the cache when fetching URLs. A client that determines that a potentially-valid certificate chain is actually invalid due to an expired OCSP response MAY retry with "forceFetch" set to retrieve an updated OCSP from the original server.

1. Let:

   * "signature" be the signature (byte sequence in the parameterised identifier's "sig" parameter).

   * "integrity" be the signature's "integrity" parameter.

   * "validity-url" be the signature's "validity-url" parameter.

   * "cert-url" be the signature's "cert-url" parameter, if any.

   * "cert-sha256" be the signature's "cert-sha256" parameter, if any.

   * "ed25519key" be the signature's "ed25519key" parameter, if any.

   * "date" be the signature's "date" parameter, interpreted as a Unix time.

   * "expires" be the signature's "expires" parameter, interpreted as a Unix time.

2. Set "publicKey" and "signing-alg" depending on which key fields are present:

   1. If "cert-url" is present:

      1. Let "certificate-chain" be the result of loading the certificate chain at "cert-url" passing the "forceFetch" flag (Section 3.3). If this returns "invalid", return "invalid".

      2. Let "main-certificate" be the first certificate in "certificate-chain".

      3. Set "publicKey" to "main-certificate"'s public key.

      4. If "publicKey" is an RSA key, return "invalid".

5. If "publicKey" is a key using the secp256r1 elliptic curve, set "signing-alg" to ecdsa_secp256r1_sha256 as defined in Section 4.2.3 of [RFC8446].

6. Otherwise, either return "invalid" or set "signing-alg" to a non-legacy signing algorithm defined by TLS 1.3 or later ([RFC8446]). This choice MUST depend only on "publicKey"'s type and not on any other context.

2. If "ed25519key" is present, set "publicKey" to "ed25519key" and "signing-alg" to ed25519, as defined by [RFC8032]

3. If "expires" is more than 7 days (604800 seconds) after "date", return "invalid".

4. If the current time is before "date" or after "expires", return "invalid".

5. Let "message" be the concatenation of the following byte strings. This matches the [RFC8446] format to avoid cross-protocol attacks if anyone uses the same key in a TLS certificate and an exchange-signing certificate.

   1. A string that consists of octet 32 (0x20) repeated 64 times.

   2. A context string: the ASCII encoding of "HTTP Exchange 1".

      Note: RFC EDITOR PLEASE DELETE THIS NOTE; The implementation of the final RFC MUST use this context string, but implementations of drafts MUST NOT use it and MUST use another draft-specific string beginning with "HTTP Exchange 1 " instead. This ensures that signers can predict how their signatures will be used.

   3. A single 0 byte which serves as a separator.

   4. If "cert-sha256" is set, a byte holding the value 32 followed by the 32 bytes of the value of "cert-sha256". Otherwise a 0 byte.

   5. The 8-byte big-endian encoding of the length in bytes of "validity-url", followed by the bytes of "validity-url".

   6. The 8-byte big-endian encoding of "date".

   7. The 8-byte big-endian encoding of "expires".

8. The 8-byte big-endian encoding of the length in bytes of "requestUrl", followed by the bytes of "requestUrl".

9. The 8-byte big-endian encoding of the length in bytes of "responseHeaders", followed by the bytes of "responseHeaders".

6. If "cert-url" is present and the SHA-256 hash of "main-certificate"'s "cert_data" is not equal to "cert-sha256" (whose presence was checked when the "Signature" header field was parsed), return "invalid".

   Note that this intentionally differs from TLS 1.3, which signs the entire certificate chain in its Certificate Verify (Section 4.4.3 of [RFC8446]), in order to allow updating the stapled OCSP response without updating signatures at the same time.

7. If "signature" is not a valid signature of "message" by "publicKey" using "signing-alg", return "invalid".

8. If "headers", interpreted according to Section 3.2, does not contain a "Content-Type" response header field (Section 3.1.1.5 of [RFC7231]), return "invalid".

   Clients MUST interpret the signed payload as this specified media type instead of trying to sniff a media type from the bytes of the payload, for example by attaching an "X-Content-Type-Options: nosniff" header field ([FETCH]) to the extracted response.

9. If "integrity" names a header field and parameter that is not present in "responseHeaders" or which the client cannot use to check the integrity of "payload" (for example, the header field is new and hasn't been implemented yet), then return "invalid". If the selected header field provides integrity guarantees weaker than SHA-256, return "invalid". If validating integrity using the selected header field requires the client to process records larger than 16384 bytes, return "invalid". Clients MUST implement at least the "Digest" header field with its "mi-sha256" digest algorithm (Section 3 of [I-D.thomson-http-mice]).

   Note: RFC EDITOR PLEASE DELETE THIS NOTE; Implementations of drafts of this RFC MUST recognize the draft spelling of the content encoding and digest algorithm specified by [I-D.thomson-http-mice] until that draft is published as an RFC. For example, implementations of draft-thomson-http-mice-03 would use "mi-sha256-03" and MUST NOT use "mi-sha256" itself. This ensures that final implementations don't need to handle compatibility with implementations of early drafts of that content encoding.

If "payload" doesn't match the integrity information in the header described by "integrity", return "invalid".

10. Return "potentially-valid" with whichever is present of "certificate-chain" or "ed25519key".

Note that the above algorithm can determine that an exchange's headers are potentially-valid before the exchange's payload is received. Similarly, if "integrity" identifies a header field and parameter like "Digest:mi-sha256" ([I-D.thomson-http-mice]) that can incrementally validate the payload, early parts of the payload can be determined to be potentially-valid before later parts of the payload. Higher-level protocols MAY process parts of the exchange that have been determined to be potentially-valid as soon as that determination is made but MUST NOT process parts of the exchange that are not yet potentially-valid. Similarly, as the higher-level protocol determines that parts of the exchange are actually valid, the client MAY process those parts of the exchange and MUST wait to process other parts of the exchange until they too are determined to be valid.

3.5.1. Open Questions

Should the signed message use the TLS format (with an initial 64 spaces) even though these certificates can't be used in TLS servers?

3.6. Updating signature validity

Both OCSP responses and signatures are designed to expire a short time after they're signed, so that revoked certificates and signed exchanges with known vulnerabilities are distrusted promptly.

This specification provides no way to update OCSP responses by themselves. Instead, clients need to re-fetch the "cert-url" (Section 3.5, Paragraph 6) to get a chain including a newer OCSP response.

The "validity-url" parameter (Section 3.1) of the signatures provides a way to fetch new signatures or learn where to fetch a complete updated exchange.

Each version of a signed exchange SHOULD have its own validity URLs, since each version needs different signatures and becomes obsolete at different times.

The resource at a "validity-url" is "validity data", a CBOR map matching the following CDDL ([CDDL]):

```
validity = {
  ? signatures: [ + bytes ]
  ? update: {
    ? size: uint,
  }
]
```

The elements of the "signatures" array are parameterised identifiers (Section 4.2.6 of [I-D.ietf-httpbis-header-structure]) meant to replace the signatures within the "Signature" header field pointing to this validity data. If the signed exchange contains a bug severe enough that clients need to stop using the content, the "signatures" array MUST NOT be present.

If the the "update" map is present, that indicates that a new version of the signed exchange is available at its effective request URI (Section 5.5 of [RFC7230]) and can give an estimate of the size of the updated exchange ("update.size"). If the signed exchange is currently the most recent version, the "update" SHOULD NOT be present.

If both the "signatures" and "update" fields are present, clients can use the estimated size to decide whether to update the whole resource or just its signatures.

3.6.1. Examples

For example, say a signed exchange whose URL is "https://example.com/resource" has the following "Signature" header field (with line breaks included and irrelevant fields omitted for ease of reading).

```
Signature:
 sig1;
  sig=*MEUCIQ...*;
  ...
  validity-url="https://example.com/resource.validity.1511157180";
  cert-url="https://example.com/oldcerts";
  date=1511128380; expires=1511733180,
 sig2;
  sig=*MEQCIG...*;
  ...
  validity-url="https://example.com/resource.validity.1511157180";
  cert-url="https://example.com/newcerts";
  date=1511128380; expires=1511733180,
 thirdpartysig;
  sig=*MEYCIQ...*;
  ...
  validity-url="https://thirdparty.example.com/resource.validity.1511161860";
  cert-url="https://thirdparty.example.com/certs";
  date=1511478660; expires=1511824260
```

At 2017-11-27 11:02 UTC, "sig1" and "sig2" have expired, but "thirdpartysig" doesn’t exipire until 23:11 that night, so the client needs to fetch "https://example.com/resource.validity.1511157180" (the "validity-url" of "sig1" and "sig2") if it wishes to update those signatures. This URL might contain:


```
{
  "signatures": [
    'sig1; '
    'sig=*MEQCIC/I9Q+7BZFP6cSDsWx43pBAL0ujTbON/+7RwKVk+ba5AiB3FSFLZqpzmDJ0NumNwN0
4pqgJZE99fcK86UjkPbj4jw==*; '
    'validity-url="https://example.com/resource.validity.1511157180"; '
    'integrity="digest/mi-sha256"; '
    'cert-url="https://example.com/newcerts"; '
    'cert-sha256=*J/lEm9kNRODdCmINbvitpvdYKNQ+YgBj99DlYp4fEXw=*; '
    'date=1511733180; expires=1512337980'
  ],
  "update": {
    "size": 5557452
  }
}
```

This indicates that the client could fetch a newer version at "https://example.com/resource" (the original URL of the exchange), or that the validity period of the old version can be extended by replacing the first two of the original signatures (the ones with a validity-url of "https://example.com/resource.validity.1511157180") with the single new signature provided. (This might happen at the end of a migration to a new root certificate.) The signatures of the updated signed exchange would be:

```
Signature:
 sig1;
  sig=*MEQCIC...*;
  ...
  validity-url="https://example.com/resource.validity.1511157180";
  cert-url="https://example.com/newcerts";
  date=1511733180; expires=1512337980,
 thirdpartysig;
  sig=*MEYCIQ...*;
  ...
  validity-url="https://thirdparty.example.com/resource.validity.1511161860";
  cert-url="https://thirdparty.example.com/certs";
  date=1511478660; expires=1511824260
```

"https://example.com/resource.validity.1511157180" could also expand the set of signatures if its "signatures" array contained more than 2 elements.

3.7. The Accept-Signature header

"Signature" header fields cost on the order of 300 bytes for ECDSA signatures, so servers might prefer to avoid sending them to clients that don't intend to use them. A client can send the "Accept-Signature" header field to indicate that it does intend to take advantage of any available signatures and to indicate what kinds of signatures it supports.

When a server receives an "Accept-Signature" header field in a client request, it SHOULD reply with any available "Signature" header fields for its response that the "Accept-Signature" header field indicates the client supports. However, if the "Accept-Signature" value violates a requirement in this section, the server MUST behave as if it hadn't received any "Accept-Signature" header at all.

The "Accept-Signature" header field is a Structured Header as defined by [I-D.ietf-httpbis-header-structure]. Its value MUST be a parameterised list (Section 3.4 of [I-D.ietf-httpbis-header-structure]). Its ABNF is:

Accept-Signature = sh-param-list

The order of identifiers in the "Accept-Signature" list is not significant. Identifiers, ignoring any initial "-" character, MUST NOT be duplicated.

Each identifier in the "Accept-Signature" header field's value indicates that a feature of the "Signature" header field (Section 3.1) is supported. If the identifier begins with a "-" character, it instead indicates that the feature named by the rest of the identifier is not supported. Unknown identifiers and parameters MUST be ignored because new identifiers and new parameters on existing identifiers may be defined by future specifications.

3.7.1. Integrity identifiers

Identifiers starting with "digest/" indicate that the client supports the "Digest" header field ([RFC3230]) with the parameter from the HTTP Digest Algorithm Values Registry (https://www.iana.org/assignments/http-dig-alg/http-dig-alg.xhtml) registry named in lower-case by the rest of the identifier. For example, "digest/mi-blake2" indicates support for Merkle integrity with the as-yet-unspecified mi-blake2 parameter, and "-digest/mi-sha256" indicates non-support for Merkle integrity with the mi-sha256 content encoding.

If the "Accept-Signature" header field is present, servers SHOULD assume support for "digest/mi-sha256" unless the header field states otherwise.

3.7.2. Key type identifiers

Identifiers starting with "ecdsa/" indicate that the client supports certificates holding ECDSA public keys on the curve named in lower-case by the rest of the identifier.

If the "Accept-Signature" header field is present, servers SHOULD assume support for "ecdsa/secp256r1" unless the header field states otherwise.

3.7.3. Key value identifiers

The "ed25519key" identifier has parameters indicating the public keys that will be used to validate the returned signature. Each parameter's name is re-interpreted as a byte sequence (Section 3.10 of [I-D.ietf-httpbis-header-structure]) encoding a prefix of the public key. For example, if the client will validate signatures using the public key whose base64 encoding is

"11qYAYKxCrfVS/7TyWQHOg7hcvPapiMlrwIaaPcHURo=", valid "Accept-Signature" header fields include:

```
Accept-Signature: ..., ed25519key; *11qYAYKxCrfVS/7TyWQHOg7hcvPapiMlrwIaaPcHURo=*
Accept-Signature: ..., ed25519key; *11qYAYKxCrfVS/7TyWQHOg==*
Accept-Signature: ..., ed25519key; *11qYAQ==*
Accept-Signature: ..., ed25519key; **
```

but not

```
Accept-Signature: ..., ed25519key; *11qYA===*
```

because 5 bytes isn't a valid length for encoded base64, and not

```
Accept-Signature: ..., ed25519key; 11qYAQ
```

because it doesn't start or end with the "*"s that indicate a byte sequence.

Note that "ed25519key; **" is an empty prefix, which matches all public keys, so it's useful in subresource integrity (Appendix A.3) cases like "<link rel=preload as=script href="...">" where the public key isn't known until the matching "<script src="..." integrity="...">" tag.

3.7.4. Examples

```
Accept-Signature: digest/mi-sha256
```

states that the client will accept signatures with payload integrity assured by the "Digest" header and "mi-sha256" digest algorithm and implies that the client will accept signatures from ECDSA keys on the secp256r1 curve.

```
Accept-Signature: -ecdsa/secp256r1, ecdsa/secp384r1
```

states that the client will accept ECDSA keys on the secp384r1 curve but not the secp256r1 curve and payload integrity assured with the "Digest: mi-sha256" header field.

3.7.5. Open Questions

Is an "Accept-Signature" header useful enough to pay for itself? If clients wind up sending it on most requests, that may cost more than the cost of sending "Signature"s unconditionally. On the other hand, it gives servers an indication of which kinds of signatures are supported, which can help us upgrade the ecosystem in the future.

Is "Accept-Signature" the right spelling, or do we want to imitate "Want-Digest" (Section 4.3.1 of [RFC3230]) instead?

Do I have the right structure for the identifiers indicating feature support?

4. Cross-origin trust

To determine whether to trust a cross-origin exchange, the client takes a "Signature" header field (Section 3.1) and the exchange's

* "requestUrl", a byte sequence that can be parsed into the exchange's effective request URI (Section 5.5 of [RFC7230]),

* "responseHeaders", a byte sequence holding the canonical serialization (Section 3.4) of the CBOR representation (Section 3.2) of the exchange's response metadata and headers, and

* "payload", a stream of bytes constituting the exchange's payload body (Section 3.3 of [RFC7230]).

The client MUST parse the "Signature" header into a list of signatures according to the instructions in Section 3.5, and run the following algorithm for each signature, stopping at the first one that returns "valid". If any signature returns "valid", return "valid". Otherwise, return "invalid".

1. If the signature's "validity-url" parameter (Section 3.1) is not same-origin (https://html.spec.whatwg.org/multipage/origin.html#same-origin) with "requestUrl", return "invalid".

2. Use Section 3.5 to determine the signature's validity for "requestUrl", "responseHeaders", and "payload", getting "certificate-chain" back. If this returned "invalid" or didn't return a certificate chain, return "invalid".

3. Let "response" be the response metadata and headers parsed out of "responseHeaders".

4. If Section 3 of [RFC7234] forbids a shared cache from storing "response", return "invalid".

5. If "response"'s headers contain an uncached header field, as defined in Section 4.1, return "invalid".

6. Let "authority" be the host component of "requestUrl".

7. Validate the "certificate-chain" using the following substeps. If any of them fail, re-run Section 3.5 once over the signature with the "forceFetch" flag set, and restart from step 2. If a substep fails again, return "invalid".

   1. Use "certificate-chain" to validate that its first entry, "main-certificate" is trusted as "authority"'s server certificate ([RFC5280] and other undocumented conventions). Let "path" be the path that was used from the "main-certificate" to a trusted root, including the "main-certificate" but excluding the root.

   2. Validate that "main-certificate" has the CanSignHttpExchanges extension (Section 4.2).

   3. Validate that "main-certificate" has an "ocsp" property (Section 3.3) with a valid OCSP response whose lifetime ("nextUpdate - thisUpdate") is less than 7 days ([RFC6960]). Note that this does not check for revocation of intermediate certificates, and clients SHOULD implement another mechanism for that.

   4. Validate that valid SCTs from trusted logs are available from any of:

      * The "SignedCertificateTimestampList" in "main-certificate"'s "sct" property (Section 3.3),

      * An OCSP extension in the OCSP response in "main-certificate"'s "ocsp" property, or

      * An X.509 extension in the certificate in "main-certificate"'s "cert" property,

      as described by Section 3.3 of [RFC6962].

8. Return "valid".

4.1. Uncached header fields

Hop-by-hop and other uncached headers MUST NOT appear in a signed exchange. These will eventually be listed in [I-D.ietf-httpbis-cache], but for now they're listed here:

* Hop-by-hop header fields listed in the Connection header field (Section 6.1 of [RFC7230]).

* Header fields listed in the no-cache response directive in the Cache-Control header field (Section 5.2.2.2 of [RFC7234]).

* Header fields defined as hop-by-hop:

  - Connection

  - Keep-Alive

  - Proxy-Connection

  - Trailer

  - Transfer-Encoding

  - Upgrade

* Stateful headers as defined below.

4.1.1. Stateful header fields

As described in Section 6.1, a publisher can cause problems if they sign an exchange that includes private information. There's no way for a client to be sure an exchange does or does not include private information, but header fields that store or convey stored state in the client are a good sign.

A stateful response header field modifies state, including authentication status, in the client. The HTTP cache is not considered part of this state. These include but are not limited to:

* "Authentication-Control", [RFC8053]

* "Authentication-Info", [RFC7615]

* "Clear-Site-Data", [W3C.WD-clear-site-data-20171130]

* "Optional-WWW-Authenticate", [RFC8053]

* "Proxy-Authenticate", [RFC7235]

* "Proxy-Authentication-Info", [RFC7615]

* "Public-Key-Pins", [RFC7469]

* "Sec-WebSocket-Accept", [RFC6455]

* "Set-Cookie", [RFC6265]

* "Set-Cookie2", [RFC2965]

* "SetProfile", [W3C.NOTE-OPS-OverHTTP]

* "Strict-Transport-Security", [RFC6797]

* "WWW-Authenticate", [RFC7235]

4.2. Certificate Requirements

We define a new X.509 extension, CanSignHttpExchanges to be used in the certificate when the certificate permits the usage of signed exchanges. When this extension is not present the client MUST NOT accept a signature from the certificate as proof that a signed exchange is authoritative for a domain covered by the certificate. When it is present, the client MUST follow the validation procedure in Section 4.

```
id-ce-canSignHttpExchanges OBJECT IDENTIFIER ::= { TBD }

CanSignHttpExchanges ::= NULL
```

Note that this extension contains an ASN.1 NULL (bytes "05 00") because some implementations have bugs with empty extensions.

Leaf certificates without this extension need to be revoked if the private key is exposed to an unauthorized entity, but they generally don’t need to be revoked if a signing oracle is exposed and then removed.

CA certificates, by contrast, need to be revoked if an unauthorized entity is able to make even one unauthorized signature.

Certificates with this extension MUST be revoked if an unauthorized entity is able to make even one unauthorized signature.

Certificates with this extension MUST have a Validity Period no greater than 90 days.

Conforming CAs MUST NOT mark this extension as critical.

A conforming CA MUST NOT issue certificates with this extension unless, for each dNSName in the subjectAltName extension of the certificate to be issued:

1. An "issue" or "issuewild" CAA property ([RFC6844]) exists that authorizes the CA to issue the certificate; and

2. The "cansignhttpexchanges" parameter (Section 4.2.1) is present on the property and is equal to "yes"

Clients MUST NOT accept certificates with this extension in TLS connections (Section 4.4.2.2 of [RFC8446]).

RFC EDITOR PLEASE DELETE THE REST OF THE PARAGRAPHS IN THIS SECTION

```
id-ce-google OBJECT IDENTIFIER ::= { 1 3 6 1 4 1 11129 }
id-ce-canSignHttpExchangesDraft OBJECT IDENTIFIER ::= { id-ce-google 2 1 22 }
```

Implementations of drafts of this specification MAY recognize the "id-ce-canSignHttpExchangesDraft" OID as identifying the CanSignHttpExchanges extension. This OID might or might not be used as the final OID for the extension, so certificates including it might need to be reissued once the final RFC is published.

Some certificates have already been issued with this extension and with validity periods longer than 90 days. These certificates will not immediately be treated as invalid. Instead:

* Clients MUST reject certificates with this extension that were issued after 2019-05-01 and have a Validity Period longer than 90 days.

* After 2019-08-01, clients MUST reject all certificates with this extension that have a Validity Period longer than 90 days.

The above requirements on CAs to limit the Validity Period and check for a CAA parameter are effective starting 2019-05-01.

4.2.1. Extensions to the CAA Record: cansignhttpexchanges Parameter

A CAA parameter "cansignhttpexchanges" is defined for the "issue" and "issuewild" properties defined by [RFC6844]. The value of this parameter, if specified, MUST be "yes".

5. Transferring a signed exchange

A signed exchange can be transferred in several ways, of which three are described here.

5.1. Same-origin response

The signature for a signed exchange can be included in a normal HTTP response. Because different clients send different request header fields, clients don't know how the server's content negotiation algorithm works, and intermediate servers add response header fields, it can be impossible to have a signature for the exchange's exact request, content negotiation, and response. Therefore, when a client calls the validation procedure in Section 3.5) to validate the "Signature" header field for an exchange represented as a normal HTTP request/response pair, it MUST pass:

* The "Signature" header field,

* The effective request URI (Section 5.5 of [RFC7230]) of the request,

* The serialized headers defined by Section 5.1.1, and

* The response's payload.

If the client relies on signature validity for any aspect of its behavior, it MUST ignore any header fields that it didn't pass to the validation procedure.

If the signed response includes a "Variants" header field, the client MUST use the cache behavior algorithm in Section 4 of [I-D.ietf-httpbis-variants] to check that the signed response is an appropriate representation for the request the client is trying to fulfil. If the response is not an appropriate representation, the client MUST treat the signature as invalid.

5.1.1. Serialized headers for a same-origin response

The serialized headers of an exchange represented as a normal HTTP request/response pair (Section 2.1 of [RFC7230] or Section 8.1 of [RFC7540]) are the canonical serialization (Section 3.4) of the CBOR representation (Section 3.2) of the response status code (Section 6 of [RFC7231]) and the response header fields whose names are listed in that response's "Signed-Headers" header field (Section 5.1.2). If a response header field name from "Signed-Headers" does not appear in the response's header fields, the exchange has no serialized headers.

If the exchange's "Signed-Headers" header field is not present, doesn't parse as a Structured Header ([I-D.ietf-httpbis-header-structure]) or doesn't follow the constraints on its value described in Section 5.1.2, the exchange has no serialized headers.

5.1.1.1. Open Questions

Do the serialized headers of an exchange need to include the "Signed-Headers" header field itself?

5.1.2. The Signed-Headers Header

The "Signed-Headers" header field identifies an ordered list of response header fields to include in a signature. The request URL and response status are included unconditionally. This allows a TLS-terminating intermediate to reorder headers without breaking the signature. This _can_ also allow the intermediate to add headers that will be ignored by some higher-level protocols, but Section 3.5 provides a hook to let other higher-level protocols reject such insecure headers.

This header field appears once instead of being incorporated into the signatures' parameters because the signed header fields need to be consistent across all signatures of an exchange, to avoid forcing higher-level protocols to merge the header field lists of valid signatures.

"Signed-Headers" is a Structured Header as defined by [I-D.ietf-httpbis-header-structure]. Its value MUST be a list (Section 3.2 of [I-D.ietf-httpbis-header-structure]). Its ABNF is:

```
Signed-Headers = sh-list
```

Each element of the "Signed-Headers" list must be a lowercase string (Section 3.8 of [I-D.ietf-httpbis-header-structure]) naming an HTTP response header field. Pseudo-header field names (Section 8.1.2.1 of [RFC7540]) MUST NOT appear in this list.

Higher-level protocols SHOULD place requirements on the minimum set of headers to include in the "Signed-Headers" header field.

5.2. HTTP/2 extension for cross-origin Server Push

To allow servers to Server-Push (Section 8.2 of [RFC7540]) signed exchanges (Section 3) signed by an authority for which the server is not authoritative (Section 9.1 of [RFC7230]), this section defines an HTTP/2 extension.

5.2.1. Indicating support for cross-origin Server Push

Clients that might accept signed Server Pushes with an authority for which the server is not authoritative indicate this using the HTTP/2 SETTINGS parameter ENABLE_CROSS_ORIGIN_PUSH (0xSETTING-TBD).

An ENABLE_CROSS_ORIGIN_PUSH value of 0 indicates that the client does not support cross-origin Push. A value of 1 indicates that the client does support cross-origin Push.

A client MUST NOT send a ENABLE_CROSS_ORIGIN_PUSH setting with a value other than 0 or 1 or a value of 0 after previously sending a value of 1. If a server receives a value that violates these rules, it MUST treat it as a connection error (Section 5.4.1 of [RFC7540]) of type PROTOCOL_ERROR.

The use of a SETTINGS parameter to opt-in to an otherwise incompatible protocol change is a use of "Extending HTTP/2" defined by Section 5.5 of [RFC7540]. If a server were to send a cross-origin Push without first receiving a ENABLE_CROSS_ORIGIN_PUSH setting with the value of 1 it would be a protocol violation.

5.2.2. NO_TRUSTED_EXCHANGE_SIGNATURE error code

The signatures on a Pushed cross-origin exchange may be untrusted for several reasons, for example that the certificate could not be fetched, that the certificate does not chain to a trusted root, that the signature itself doesn’t validate, that the signature is expired, etc. This draft conflates all of these possible failures into one error code, NO_TRUSTED_EXCHANGE_SIGNATURE (0xERROR-TBD).

5.2.2.1. Open Questions

How fine-grained should this specification’s error codes be?

5.2.3. Validating a cross-origin Push

If the client has set the ENABLE_CROSS_ORIGIN_PUSH setting to 1, the server MAY Push a signed exchange for which it is not authoritative, and the client MUST NOT treat a PUSH_PROMISE for which the server is not authoritative as a stream error (Section 5.4.2 of [RFC7540]) of type PROTOCOL_ERROR, as described in Section 8.2 of [RFC7540], unless there is another error as described below.

Instead, the client MUST validate such a PUSH_PROMISE and its response against the following list:

1. If the PUSH_PROMISE includes any non-pseudo request header fields, the client MUST treat it as a stream error (Section 5.4.2 of [RFC7540]) of type PROTOCOL_ERROR.

2. If the PUSH_PROMISE’s method is not "GET", the client MUST treat it as a stream error (Section 5.4.2 of [RFC7540]) of type PROTOCOL_ERROR.

3. Run the algorithm in Section 4 over:

   * The "Signature" header field from the response.

   * The effective request URI from the PUSH_PROMISE.

   * The canonical serialization (Section 3.4) of the CBOR representation (Section 3.2) of the pushed response's status and its headers except for the "Signature" header field.

   * The response's payload.

   If this returns "invalid", the client MUST treat the response as a stream error (Section 5.4.2 of [RFC7540]) of type NO_TRUSTED_EXCHANGE_SIGNATURE. Otherwise, the client MUST treat the pushed response as if the server were authoritative for the PUSH_PROMISE's authority.

5.2.3.1. Open Questions

Is it right that "validity-url" is required to be same-origin with the exchange? This allows the mitigation against downgrades in Section 6.3, but prohibits intermediates from providing a cache of the validity information. We could do both with a list of URLs.

5.3. application/signed-exchange format

To allow signed exchanges to be the targets of "<link rel=prefetch>" tags, we define the "application/signed-exchange" content type that represents a signed HTTP exchange, including a request URL, response metadata and header fields, and a response payload.

When served over HTTP, a response containing an "application/signed-exchange" payload MUST include at least the following response header fields, to reduce content sniffing vulnerabilities (Section 6.8):

* Content-Type: application/signed-exchange;v=_version_

* X-Content-Type-Options: nosniff

This content type consists of the concatenation of the following items:

1. 8 bytes consisting of the ASCII characters "sxg1" followed by 4 0x00 bytes, to serve as a file signature. This is redundant with the MIME type, and recipients that receive both MUST check that they match and stop parsing if they don't.

Note: RFC EDITOR PLEASE DELETE THIS NOTE; The implementation of the final RFC MUST use this file signature, but implementations of drafts MUST NOT use it and MUST use another implementation-specific 8-byte string beginning with "sxg1-".

2. 2 bytes storing a big-endian integer "fallbackUrlLength".

3. "fallbackUrlLength" bytes holding a "fallbackUrl", which MUST UTF-8 decode to an absolute URL with a scheme of "https".

   Note: The byte location of the fallback URL is intended to remain invariant across versions of the "application/signed-exchange" format so that parsers encountering unknown versions can always find a URL to redirect to.

   Issue: Should this fallback information also include the method?

4. 3 bytes storing a big-endian integer "sigLength". If this is larger than 16384 (16*1024), parsing MUST fail.

5. 3 bytes storing a big-endian integer "headerLength". If this is larger than 524288 (512*1024), parsing MUST fail.

6. "sigLength" bytes holding the "Signature" header field's value (Section 3.1).

7. "headerLength" bytes holding "signedHeaders", the canonical serialization (Section 3.4) of the CBOR representation of the response headers of the exchange represented by the "application/signed-exchange" resource (Section 3.2), excluding the "Signature" header field.

8. The payload body (Section 3.3 of [RFC7230]) of the exchange represented by the "application/signed-exchange" resource.

   Note that the use of the payload body here means that a "Transfer-Encoding" header field inside the "application/signed-exchange" header block has no effect. A "Transfer-Encoding" header field on the outer HTTP response that transfers this resource still has its normal effect.

5.3.1. Cross-origin trust in application/signed-exchange

To determine whether to trust a cross-origin exchange stored in an "application/signed-exchange" resource, pass the "Signature" header field's value, "fallbackUrl" as the effective request URI, "signedHeaders", and the payload body to the algorithm in Section 4.

5.3.2. Example

An example "application/signed-exchange" file representing a possible signed exchange with https://example.com/ (https://example.com/) follows, with lengths represented by descriptions in "<>"s, CBOR represented in the extended diagnostic format defined in Appendix G of [CDDL], and most of the "Signature" header field and payload elided with a ...:

```
sxg1\0\0\0\0<2-byte length of the following url string>
https://example.com/<3-byte length of the following header
value><3-byte length of the encoding of the
following map>sig1; sig=*...; integrity="digest/mi-sha256"; ...{
  ':status': '200',
  'content-type': 'text/html'
}<!doctype html>\r\n<html>...
```

5.3.3. Open Questions

Should this be a CBOR format, or is the current mix of binary and CBOR better?

Are the mime type, extension, and magic number right?

6. Security considerations

6.1. Over-signing

If a publisher blindly signs all responses as their origin, they can cause at least two kinds of problems, described below. To avoid this, publishers SHOULD design their systems to opt particular public content that doesn't depend on authentication status into signatures instead of signing by default.

Signing systems SHOULD also incorporate the following mitigations to reduce the risk that private responses are signed:

1. Strip the "Cookie" request header field and other identifying information like client authentication and TLS session IDs from requests whose exchange is destined to be signed, before forwarding the request to a backend.

2. Only sign exchanges where the response includes a "Cache-Control: public" header. Clients are not required to fail signature-checking for exchanges that omit this "Cache-Control" response header field to reduce the risk that naive signing systems blindly add it.

6.1.1. Session fixation

Blind signing can sign responses that create session cookies or otherwise change state on the client to identify a particular session. This breaks certain kinds of CSRF defense and can allow an attacker to force a user into the attacker’s account, where the user might unintentionally save private information, like credit card numbers or addresses.

This specification defends against cookie-based attacks by blocking the "Set-Cookie" response header, but it cannot prevent Javascript or other response content from changing state.

6.1.2. Misleading content

If a site signs private information, an attacker might set up their own account to show particular private information, forward that signed information to a victim, and use that victim’s confusion in a more sophisticated attack.

Stripping authentication information from requests before sending them to backends is likely to prevent the backend from showing attacker-specific information in the signed response. It does not prevent the attacker from showing their victim a signed-out page when the victim is actually signed in, but while this is still misleading, it seems less likely to be useful to the attacker.

6.2. Off-path attackers

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 or IP routing, they need only convince the user to visit another Web site in order to serve responses signed as the target. This consideration and mitigations for it are shared by the combination of [RFC8336] and [I-D.ietf-httpbis-http2-secondary-certs], and are discussed further in [I-D.bishop-httpbis-origin-fed-up].

6.2.1. Mis-issued certificates

If a CA mis-issues a certificate for a domain, this specification provides a way to detect the mis-issuance and mitigate harm within approximately two weeks. Specifically, because all signed exchanges must include a "SignedCertificateTimestampList" ([RFC6962], a CT log has promised to publish the mis-issued certificate within that log’s Maximum Merge Delay, 1 day for many logs. The domain owner can then detect the mis-issued certificate and notify the CA to revoke it, which the [BRs], section 4.9.1.1, say they must do within another 5 days.

Once the mis-issued certificate is revoked, existing OCSP responses begin to expire. The [BRs], section 4.9.10, require that OCSP responses have a maximum expiration time of 10 days, after which they can't be used to validate a certificate chain (Section 3.3). This leads to a total compromised time of 16 days after a mis-issuance.

However, CAs might future-date their OCSP responses, in which case the mitigation doesn't work.

CAs are forbidden from future-dating their OCSP responses by the [BRs] section 4.9.9, "OCSP responses MUST conform to RFC6960 and/or RFC5019." [RFC6960] includes, "The time at which the status was known to be correct SHALL be reflected in the thisUpdate field of the response.", and [RFC5019] includes, "When pre-producing OCSPResponse messages, the responder MUST set the thisUpdate, nextUpdate, and producedAt times as follows: thisUpdate: The time at which the status being indicated is known to be correct."

However, if a CA violates the [BRs] to sign future-dated OCSP responses, attempts to keep the nonconformant OCSP responses private, but then leaks them, it could cause clients to trust a hostile signed exchange long after its certificate has been revoked.

Clients could use systems like [CRLSets] and [OneCrl] to revoke the intermediate certificate that signed the future-dated OCSP responses.

6.2.2. Stolen private keys

If the private key for a CanSignHttpExchanges certificate is stolen, it can be used at scale until the certificate expires or is revoked, and unlike for a stolen key for a normal TLS-terminating certificate, the rightful owner can't detect the problem by watching for attacks on the DNS or routing infrastructure.

This specification does not currently propose a way for the rightful owner to detect that their keys are being used by an attacker, after they've opted into the risk by requesting a CanSignHttpExchanges certificate in the first place. Clients can fetch a signature's "validity-url" (Section 3.1) to help owners detect key compromise, but that compromises some of the privacy properties of this specification.

6.3. Downgrades

Signing a bad response can affect more users than simply serving a bad response, since a served response will only affect users who make a request while the bad version is live, while an attacker can forward a signed response until its signature expires. Publishers should consider shorter signature expiration times than they use for cache expiration times.

Clients MAY also check the "validity-url" (Section 3.1) of an exchange more often than the signature's expiration would require. Doing so for an exchange with an HTTPS request URI provides a TLS guarantee that the exchange isn't out of date (as long as Section 5.2.3.1 is resolved to keep the same-origin requirement).

6.4. Signing oracles are permanent

An attacker with temporary access to a signing oracle can sign "still valid" assertions with arbitrary timestamps and expiration times. As a result, when a signing oracle is removed, the keys it provided access to MUST be revoked so that, even if the attacker used them to sign future-dated exchange validity assertions, the key's OCSP assertion will expire, causing the exchange as a whole to become untrusted.

6.5. Unsigned headers

The use of a single "Signed-Headers" header field prevents us from signing aspects of the request other than its effective request URI (Section 5.5 of [RFC7230]). For example, if a publisher signs both "Content-Encoding: br" and "Content-Encoding: gzip" variants of a response, what's the impact if an attacker serves the brotli one for a request with "Accept-Encoding: gzip"? This is mitigated by using [I-D.ietf-httpbis-variants] instead of request headers to describe how the client should run content negotiation.

The simple form of "Signed-Headers" also prevents us from signing less than the full request URL. The SRI use case (Appendix A.3) may benefit from being able to leave the authority less constrained.

Section 3.5 can succeed when some delivered headers aren't included in the signed set. This accommodates current TLS-terminating intermediates and may be useful for SRI (Appendix A.3), but is risky for trusting cross-origin responses (Appendix A.1, Appendix A.2, and Appendix A.6). Section 5.2 requires all headers to be included in the signature before trusting cross-origin pushed resources, at Ryan Sleevi's recommendation.

6.6. application/signed-exchange

Clients MUST NOT trust an effective request URI claimed by an "application/signed-exchange" resource (Section 5.3) without either ensuring the resource was transferred from a server that was authoritative (Section 9.1 of [RFC7230]) for that URI's origin, or calling the algorithm in Section 5.3.1 and getting "valid" back.

6.7. Key re-use with TLS

In general, key re-use across multiple protocols is a bad idea.

Using an exchange-signing key in a TLS (or other directly-internet-facing) server increases the risk that an attacker can steal the private key, which will allow them to mint packages (similar to Section 6.4) until their theft is discovered.

Using a TLS key in a CanSignHttpExchanges certificate makes it less likely that the server operator will discover key theft, due to the considerations in Section 6.2.

This specification uses the CanSignHttpExchanges X.509 extension (Section 4.2) to discourage re-use of TLS keys to sign exchanges or vice-versa.

We require that clients reject certificates with the CanSignHttpExchanges extension when making TLS connections to minimize the chance that servers will re-use keys like this. Ideally, we would make the extension critical so that even clients that don't understand it would reject such TLS connections, but this proved impossible because certificate-validating libraries ship on significantly different schedules from the clients that use them.

Even once all clients reject these certificates in TLS connections, this will still just discourage and not prevent key re-use, since a server operator can unwisely request two different certificates with the same private key.

6.8. Content sniffing

While modern browsers tend to trust the "Content-Type" header sent with a resource, especially when accompanied by "X-Content-Type-Options: nosniff", plugins will sometimes search for executable content buried inside a resource and execute it in the context of the origin that served the resource, leading to XSS vulnerabilities. For example, some PDF reader plugins look for "%PDF" anywhere in the first 1kB and execute the code that follows it.

The "application/signed-exchange" format (Section 5.3) includes a URL and response headers early in the format, which an attacker could use to cause these plugins to sniff a bad content type.

To avoid vulnerabilities, in addition to the response header requirements in Section 5.3, servers are advised to only serve an "application/signed-exchange" resource (SXG) from a domain if it would also be safe for that domain to serve the SXG's content directly, and to follow at least one of the following strategies:

1. Only serve signed exchanges from dedicated domains that don't have access to sensitive cookies or user storage.

2. Generate signed exchanges "offline", that is, in response to a trusted author submitting content or existing signatures reaching a certain age, rather than in response to untrusted-reader queries.

3. Do all of:

    1. If the SXG's fallback URL (Section 5.3) is derived from the request URL, percent-encode (https://url.spec.whatwg.org/#percent-encode) ([URL]) any bytes that are greater than 0x7E or are not URL code points (https://url.spec.whatwg.org/#url-code-points) ([URL]) in the fallback URL . It is particularly important to make sure no unescaped nulls (0x00) or angle brackets (0x3C and 0x3E) appear.

    2. Do not reflect request header fields into the set of response headers.

There are still a few binary length fields that an attacker may influence to contain sensitive bytes, but they're always followed by lowercase alphabetic strings from a small set of possibilities, which reduces the chance that a client will sniff them as indicating a particular content type.

To encourage servers to include the "X-Content-Type-Options: nosniff" header field, clients SHOULD reject signed exchanges served without it.

7. Privacy considerations

7.1. Visibility of resource requests

Normally, when a client follows a link from https://source.example/page.html to "https://publisher.example/page.html", "publisher.example" learns that the client is interested in the resource. "source.example" also has several ways of discovering that the client has clicked the link, including the use of Javascript to record the click or having the link point to a URL that serves a 302 redirect to the real target.

If "publisher.example" signs "page.html" into "page.sxg", "distributor.example" serves it as "https://distributor.example/publisher/page.sxg", and the client fetches it from there, then "distributor.example" learns that the client is interested, and if the client executes some Javascript on the page or makes subresource requests, that could also report the client’s interest back to "publisher.example".

To prevent network operators other than "distributor.example" or "publisher.example" from learning which exchanges were read, clients SHOULD only load exchanges fetched over a transport that’s protected from eavesdroppers. This can be difficult to determine when the exchange is being loaded from local disk, but when the client itself requested the exchange over a network it SHOULD require TLS ([RFC8446]) or a successor transport layer, and MUST NOT accept exchanges transferred over plain HTTP without TLS.

If "source.example" and "distributor.example" are controlled by the same entity, no extra information escapes here. If they are run by different entities, a similar amount of information escapes as if "source.example" had implemented its click tracking by outsourcing to a service like https://bit.ly/ (https://bit.ly/).

There has been discussion of allowing a publisher to restrict the set of distributors that can host its signed content. If that’s added, then the privacy situation becomes more similar to the situation with CDNs, where a publisher chooses a CDN to serve their content, and the CDN learns about all requests for that content. Here the publisher would choose one or more distributors, and the distributor(s) would learn about requests for the content.

For non-executable resource types, a signed response can improve the privacy situation by hiding the client’s interest from the original publisher.

7.2. User ID transfer

If a request for "https://distributor.example/publisher/page.sxg" comes with the source's or distributor's user ID for the user, either because it's sent with the distributor's cookies or because the source stashes an encoded user ID into either the request's path or a subdomain, the distributor has a few ways to pass that user ID on to the publisher that signed the page:

1. If the distributor has the publisher's signing keys, it can sign a new page with its user ID directly embedded.

2. Otherwise, the publisher can sign lots of copies of their package, and the distributor can choose a particular copy to send a subset of the bits in its user ID to the publisher on each click, which will eventually transfer the whole thing.

To prevent this, the request for a signed exchange needs to omit credentials and block them from appearing in the URL in the same way it would block them from appearing in a cross-origin URL. We're exploring ways the link can mark the request so user agents can take the right counter-measures.

8. IANA considerations

TODO: possibly register the validity-url format.

8.1. Signature Header Field Registration

This section registers the "Signature" header field in the "Permanent Message Header Field Names" registry ([RFC3864]).

Header field name: "Signature"

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document(s): Section 3.1 of this document

8.2. Accept-Signature Header Field Registration

This section registers the "Accept-Signature" header field in the "Permanent Message Header Field Names" registry ([RFC3864]).

Header field name: "Accept-Signature"

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document(s): Section 3.7 of this document

8.3. Signed-Headers Header Field Registration

This section registers the "Signed-Headers" header field in the "Permanent Message Header Field Names" registry ([RFC3864]).

Header field name: "Signed-Headers"

Applicable protocol: http

Status: standard

Author/Change controller: IETF

Specification document(s): Section 5.1.2 of this document

8.4. HTTP/2 Settings

This section establishes an entry for the HTTP/2 Settings Registry that was established by Section 11.3 of [RFC7540]

Name: ENABLE_CROSS_ORIGIN_PUSH

Code: 0xSETTING-TBD

Initial Value: 0

Specification: This document

8.5. HTTP/2 Error code

This section establishes an entry for the HTTP/2 Error Code Registry that was established by Section 11.4 of [RFC7540]

Name: NO_TRUSTED_EXCHANGE_SIGNATURE

Code: 0xERROR-TBD

Description: The client does not trust the signature for a cross-origin Pushed signed exchange.

Specification: This document

8.6. Internet Media Type application/signed-exchange

IANA is requested to register the MIME media type ([IANA.media-types]) for signed exchanges, application/signed-exchange, as follows:

Type name: application

Subtype name: signed-exchange

Required parameters:

* v: A string denoting the version of the file format. ([RFC5234] ABNF: "version = DIGIT/%x61-7A") The version defined in this specification is "1". When used with the "Accept" header field (Section 5.3.2 of [RFC7231]), this parameter can be a comma (,)-separated list of version strings. ([RFC5234] ABNF: "version-list = version *("," version)") The server is then expected to reply with a resource using a particular version from that list.

  Note: RFC EDITOR PLEASE DELETE THIS NOTE; Implementations of drafts of this specification MUST NOT use simple integers to describe their versions, and MUST instead define implementation-specific strings to identify which draft is implemented. The newest version of [I-D.yasskin-httpbis-origin-signed-exchanges-impl] describes the meaning of one such string.

Optional parameters: N/A

Encoding considerations: binary

Security considerations: see Section 6.6

Interoperability considerations: N/A

Published specification: This specification (see Section 5.3).

Applications that use this media type: N/A

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): 73 78 67 31 00

File extension(s): .sxg

Macintosh file type code(s): N/A

Person and email address to contact for further information: See Authors' Addresses section.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See Authors' Addresses section.

Change controller: IESG

Provisional registration? Yes

8.7. Internet Media Type application/cert-chain+cbor

IANA is requested to register the MIME media type ([IANA.media-types]) for CBOR-format certificate chains, application/cert-chain+cbor, as follows:

Type name: application

Subtype name: cert-chain+cbor

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: N/A

Interoperability considerations: N/A

Published specification: This specification (see Section 3.3).

Applications that use this media type: N/A

Fragment identifier considerations: N/A

Additional information:

Deprecated alias names for this type: N/A

Magic number(s): 1*9(??) 67 F0 9F 93 9C E2 9B 93

File extension(s): N/A

Macintosh file type code(s): N/A

Person and email address to contact for further information: See Authors’ Addresses section.

Intended usage: COMMON

Restrictions on usage: N/A

Author: See Authors’ Addresses section.

Change controller: IESG

Provisional registration? Yes

8.8. The cansignhttpexchanges CAA Parameter

There are no IANA considerations for this parameter.

9. References

9.1. Normative References

[CDDL] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[FETCH] WHATWG, "Fetch", July 2020, <https://fetch.spec.whatwg.org/>.

[I-D.ietf-httpbis-header-structure] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-header-structure-19, 3 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-httpbis-header-structure-19.txt>.

[I-D.ietf-httpbis-variants] Nottingham, M., "HTTP Representation Variants", Work in Progress, Internet-Draft, draft-ietf-httpbis-variants-06, 3 November 2019, <http://www.ietf.org/internet-drafts/draft-ietf-httpbis-variants-06.txt>.

[I-D.thomson-http-mice]
Thomson, M. and J. Yasskin, "Merkle Integrity Content Encoding", Work in Progress, Internet-Draft, draft-thomson-http-mice-03, 13 August 2018, <http://www.ietf.org/internet-drafts/draft-thomson-http-mice-03.txt>.

[IANA.media-types]
IANA, "Media Types", <http://www.iana.org/assignments/media-types>.

[POSIX] IEEE and The Open Group, "The Open Group Base Specifications Issue 7", value 1003.1-2008, 2016 Edition, name IEEE, 2016, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/>.

[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>.

[RFC3230] Mogul, J. and A. Van Hoff, "Instance Digests in HTTP", RFC 3230, DOI 10.17487/RFC3230, January 2002, <https://www.rfc-editor.org/info/rfc3230>.

[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>.

[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>.

[RFC6844] Hallam-Baker, P. and R. Stradling, "DNS Certification Authority Authorization (CAA) Resource Record", RFC 6844, DOI 10.17487/RFC6844, January 2013, <https://www.rfc-editor.org/info/rfc6844>.

[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>.

[RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <https://www.rfc-editor.org/info/rfc7049>.

[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>.

[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>.

[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>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[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>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[URL] WHATWG, "URL", July 2020, <https://url.spec.whatwg.org/>.

9.2. Informative References

[BRs] CA/Browser Forum, "Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates", 10 December 2018, <https://cabforum.org/baseline-requirements-documents/>.

[CRLSets] Langley, A., "Revocation checking and Chrome's CRL", 5 February 2012, <https://www.imperialviolet.org/2012/02/05/crlsets.html>.

[I-D.bishop-httpbis-origin-fed-up] Bishop, M. and E. Nygren, "DNS Security with HTTP/2 ORIGIN", Work in Progress, Internet-Draft, draft-bishop-httpbis-origin-fed-up-00, 8 January 2019, <http://www.ietf.org/internet-drafts/draft-bishop-httpbis-origin-fed-up-00.txt>.

[I-D.burke-content-signature] Burke, B., "HTTP Header for digital signatures", Work in Progress, Internet-Draft, draft-burke-content-signature-00, 7 March 2011, <http://www.ietf.org/internet-drafts/draft-burke-content-signature-00.txt>.

[I-D.cavage-http-signatures] Cavage, M. and M. Sporny, "Signing HTTP Messages", Work in Progress, Internet-Draft, draft-cavage-http-signatures-12, 21 October 2019, <http://www.ietf.org/internet-drafts/draft-cavage-http-signatures-12.txt>.

[I-D.ietf-httpbis-cache] Fielding, R., Nottingham, M., and J. Reschke, "HTTP Caching", Work in Progress, Internet-Draft, draft-ietf-httpbis-cache-10, 12 July 2020, <http://www.ietf.org/internet-drafts/draft-ietf-httpbis-cache-10.txt>.

[I-D.ietf-httpbis-http2-secondary-certs] Bishop, M., Sullivan, N., and M. Thomson, "Secondary Certificate Authentication in HTTP/2", Work in Progress, Internet-Draft, draft-ietf-httpbis-http2-secondary-certs-06, 14 May 2020, <http://www.ietf.org/internet-drafts/draft-ietf-httpbis-http2-secondary-certs-06.txt>.

[I-D.thomson-http-content-signature]
Thomson, M., "Content-Signature Header Field for HTTP", Work in Progress, Internet-Draft, draft-thomson-http-content-signature-00, 2 July 2015, <http://www.ietf.org/internet-drafts/draft-thomson-http-content-signature-00.txt>.

[I-D.yasskin-httpbis-origin-signed-exchanges-impl]
Yasskin, J. and K. Ueno, "Signed HTTP Exchanges Implementation Checkpoints", Work in Progress, Internet-Draft, draft-yasskin-httpbis-origin-signed-exchanges-impl-03, 25 July 2019, <http://www.ietf.org/internet-drafts/draft-yasskin-httpbis-origin-signed-exchanges-impl-03.txt>.

[I-D.yasskin-wpack-use-cases]
Yasskin, J., "Use Cases and Requirements for Web Packages", Work in Progress, Internet-Draft, draft-yasskin-wpack-use-cases-00, 30 October 2019, <http://www.ietf.org/internet-drafts/draft-yasskin-wpack-use-cases-00.txt>.

[OneCrl] Goodwin, M., "Revoking Intermediate Certificates: Introducing OneCRL", 3 March 2015, <https://blog.mozilla.org/security/2015/03/03/revoking-intermediate-certificates-introducing-onecrl/>.

[RFC2965] Kristol, D. and L. Montulli, "HTTP State Management Mechanism", RFC 2965, DOI 10.17487/RFC2965, October 2000, <https://www.rfc-editor.org/info/rfc2965>.

[RFC5019] Deacon, A. and R. Hurst, "The Lightweight Online Certificate Status Protocol (OCSP) Profile for High-Volume Environments", RFC 5019, DOI 10.17487/RFC5019, September 2007, <https://www.rfc-editor.org/info/rfc5019>.

[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>.

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

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

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/info/rfc6455>.

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

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

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

[RFC7615] Reschke, J., "HTTP Authentication-Info and Proxy-Authentication-Info Response Header Fields", RFC 7615, DOI 10.17487/RFC7615, September 2015, <https://www.rfc-editor.org/info/rfc7615>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[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, <https://www.rfc-editor.org/info/rfc8053>.

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

[SRI] Akhawe, D., Braun, F., Marier, F., and J. Weinberger, "Subresource Integrity", World Wide Web Consortium Recommendation REC-SRI-20160623, 23 June 2016, <http://www.w3.org/TR/2016/REC-SRI-20160623>.

[W3C.NOTE-OPS-OverHTTP]
Hensley, P., Metral, M., Shardanand, U., Converse, D., and M. Myers, "Implementation of OPS Over HTTP", W3C NOTE NOTE-OPS-OverHTTP, 2 June 1997, <http://www.w3.org/TR/NOTE-OPS-OverHTTP>.

[W3C.WD-clear-site-data-20171130]
West, M., "Clear Site Data", World Wide Web Consortium WD WD-clear-site-data-20171130, 30 November 2017, <https://www.w3.org/TR/2017/WD-clear-site-data-20171130>.

Appendix A. Use cases

A.1. PUSHed subresources

To reduce round trips, a server might use HTTP/2 Push (Section 8.2 of [RFC7540]) to inject a subresource from another server into the client’s cache. If anything about the subresource is expired or can’t be verified, the client would fetch it from the original server.

For example, if "https://example.com/index.html" includes

```
<script src="https://jquery.com/jquery-1.2.3.min.js">
```

Then to avoid the need to look up and connect to "jquery.com" in the critical path, "example.com" might push that resource signed by "jquery.com".

A.2. Explicit use of a content distributor for subresources

In order to speed up loading but still maintain control over its content, an HTML page in a particular origin "O.com" could tell clients to load its subresources from an intermediate content distributor that’s not authoritative, but require that those resources be signed by "O.com" so that the distributor couldn’t modify the resources. This is more constrained than the common CDN case where "O.com" has a CNAME granting the CDN the right to serve arbitrary content as "O.com".

```
<img logicalsrc="https://O.com/img.png"
     physicalsrc="https://distributor.com/O.com/img.png">
```

To make it easier to configure the right distributor for a given request, computation of the "physicalsrc" could be encapsulated in a custom element:

```
<dist-img src="https://O.com/img.png"></dist-img>
```

where the "<dist-img>" implementation generates an appropriate "<img>" based on, for example, a "<meta name="dist-base">" tag elsewhere in the page. However, this has the downside that the preloader (https://calendar.perfplanet.com/2013/big-bad-preloader/) can no longer see the physical source to download it. The resulting delay might cancel out the benefit of using a distributor.

This could be used for some of the same purposes as SRI (Appendix A.3).

To implement this with the current proposal, the distributor would respond to the physical request to "https://distributor.com/O.com/img.png" with first a signed PUSH_PROMISE for "https://O.com/img.png" and then a redirect to "https://O.com/img.png".

A.3. Subresource Integrity

The W3C WebAppSec group is investigating using signatures (https://github.com/mikewest/signature-based-sri) in [SRI]. They need a way to transmit the signature with the response, which this proposal provides.

Their needs are simpler than most other use cases in that the "integrity="ed25519-[public-key]"" attribute and CSP-based ways of expressing a public key don't need that key to be wrapped into a certificate.

The "ed25519key" signature parameter supports this simpler way of attaching a key.

The current proposal for signature-based SRI describes signing only the content of a resource, while this specification requires them to sign the request URI as well. This issue is tracked in https://github.com/mikewest/signature-based-sri/issues/5 (https://github.com/mikewest/signature-based-sri/issues/5). The details of what they need to sign will affect whether and how they can use this proposal.

A.4. Binary Transparency

So-called "Binary Transparency" may eventually allow users to verify that a program they've been delivered is one that's available to the public, and not a specially-built version intended to attack just them. Binary transparency systems don't exist yet, but they're likely to work similarly to the successful Certificate Transparency logs described by [RFC6962].

Certificate Transparency depends on Signed Certificate Timestamps that prove a log contained a particular certificate at a particular time. To build the same thing for Binary Transparency logs containing HTTP resources or full websites, we'll need a way to provide signatures of those resources, which signed exchanges provides.

A.5. Static Analysis

Native app stores like the Apple App Store (https://www.apple.com/ios/app-store/) and the Android Play Store (https://play.google.com/store) grant their contents powerful abilities, which they attempt to make safe by analyzing the applications before offering them to people. The web has no equivalent way for people to wait to run an update of a web application until a trusted authority has vouched for it.

While full application analysis probably needs to wait until the authority can sign bundles of exchanges, authorities may be able to guarantee certain properties by just checking a top-level resource and its [SRI]-constrained sub-resources.

A.6. Offline websites

Fully-offline websites can be represented as bundles of signed exchanges, although an optimization to reduce the number of signature verifications may be needed. Work on this is in progress in the https://github.com/WICG/webpackage (https://github.com/WICG/webpackage) repository.

Appendix B. Requirements

B.1. Proof of origin

To verify that a thing came from a particular origin, for use in the same context as a TLS connection, we need someone to vouch for the signing key with as much verification as the signing keys used in TLS. The obvious way to do this is to re-use the web PKI and CA ecosystem.

B.1.1. Certificate constraints

If we re-use existing TLS server certificates, we incur the risks that:

1. TLS server certificates must be accessible from online servers, so they're easier to steal or use as signing oracles than an offline key. An exchange's signing key doesn't need to be online.

2. A server using an origin-trusted key for one purpose (e.g. TLS) might accidentally sign something that looks like an exchange, or vice versa.

These risks are considered too high, so we define a new X.509 certificate extension in Section 4.2 that requires CAs to issue new certificates for this purpose. We expect at least one low-cost CA to be willing to sign certificates with this extension.

B.1.2. Signature constraints

In order to prevent an attacker who can convince the server to sign some resource from causing those signed bytes to be interpreted as something else the new X.509 extension here is forbidden from being used in TLS servers. If Section 4.2 changes to allow re-use in TLS servers, we would need to:

1. Avoid key types that are used for non-TLS protocols whose output could be confused with a signature. That may be just the "rsaEncryption" OID from [RFC8017].

2. Use the same format as TLS's signatures, specified in Section 4.4.3 of [RFC8446], with a context string that's specific to this use.

The specification also needs to define which signing algorithm to use. It currently specifies that as a function from the key type, instead of allowing attacker-controlled data to specify it.

B.1.3. Retrieving the certificate

The client needs to be able to find the certificate vouching for the signing key, a chain from that certificate to a trusted root, and possibly other trust information like SCTs ([RFC6962]). One approach would be to include the certificate and its chain in the signature metadata itself, but this wastes bytes when the same certificate is used for multiple HTTP responses. If we decide to put the signature in an HTTP header, certificates are also unusually large for that context.

Another option is to pass a URL that the client can fetch to retrieve the certificate and chain. To avoid extra round trips in fetching that URL, it could be bundled (Appendix A.6) with the signed content or PUSHed (Appendix A.1) with it. The risks from the "client_certificate_url" extension (Section 11.3 of [RFC6066]) don't seem to apply here, since an attacker who can get a client to load an exchange and fetch the certificates it references, can also get the client to perform those fetches by loading other HTML.

To avoid using an unintended certificate with the same public key as the intended one, the content of the leaf certificate or the chain should be included in the signed data, like TLS does (Section 4.4.3 of [RFC8446]).

B.2. How much to sign

The previous [I-D.thomson-http-content-signature] and [I-D.burke-content-signature] schemes signed just the content, while ([I-D.cavage-http-signatures] could also sign the response headers and the request method and path. However, the same path, response headers, and content may mean something very different when retrieved from a different server. Section 5.1.1 currently includes the whole request URL in the signature, but it's possible we need a more flexible scheme to allow some higher-level protocols to accept a less-signed URL.

Servers might want to sign other request headers in order to capture their effects on content negotiation. However, there's no standard algorithm to check that a client's actual request headers match request headers sent by a server. The most promising attempt at this is [I-D.ietf-httpbis-variants], which encodes the content negotiation algorithm into the "Variants" and "Variant-Key" response headers. The proposal here (Section 3) assumes that is in use and doesn't sign request headers.

B.2.1. Conveying the signed headers

HTTP headers are traditionally munged by proxies, making it impossible to guarantee that the client will see the same sequence of bytes as the publisher published. In the HTTPS world, we have more end-to-end header integrity, but it's still likely that there are enough TLS-terminating proxies that the publisher's signatures would tend to break before getting to the client.

There's no way in current HTTP for the response to a client-initiated request (Section 8.1 of [RFC7540]) to convey the request headers it expected to respond to, but we sidestep that by conveying content negotiation information in response headers, per [I-D.ietf-httpbis-variants].

Since proxies are unlikely to modify unknown content types, we can wrap the original exchange into an "application/signed-exchange" format (Section 5.3) and include the "Cache-Control: no-transform" header when sending it.

To reduce the likelihood of accidental modification by proxies, the "application/signed-exchange" format includes a file signature that doesn't collide with other known signatures.

To help the PUSHed subresources use case (Appendix A.1), we might also want to extend the "PUSH_PROMISE" frame type to include a signature, and that could tell intermediates not to change the ensuing headers.

B.3. Response lifespan

A normal HTTPS response is authoritative only for one client, for as long as its cache headers say it should live. A signed exchange can be re-used for many clients, and if it was generated while a server was compromised, it can continue compromising clients even if their requests happen after the server recovers. This signing scheme needs to mitigate that risk.

B.3.1. Certificate revocation

Certificates are mis-issued and private keys are stolen, and in response clients need to be able to stop trusting these certificates as promptly as possible. Online revocation checks don't work (https://www.imperialviolet.org/2012/02/05/crlsets.html), so the industry has moved to pushed revocation lists and stapled OCSP responses [RFC6066].

Pushed revocation lists work as-is to block trust in the certificate signing an exchange, but the signatures need an explicit strategy to staple OCSP responses. One option is to extend the certificate download (Appendix B.1.3) to include the OCSP response too, perhaps in the TLS 1.3 CertificateEntry (https://tlswg.github.io/tls13-spec/draft-ietf-tls-tls13.html#ocsp-and-sct) format.

B.3.2. Response downgrade attacks

The signed content in a response might be vulnerable to attacks, such as XSS, or might simply be discovered to be incorrect after publication. Once the author fixes those vulnerabilities or mistakes, clients should stop trusting the old signed content in a reasonable amount of time. Similar to certificate revocation, I expect the best option to be stapled "this version is still valid" assertions with short expiration times.

These assertions could be structured as:

1. A signed minimum version number or timestamp for a set of request headers: This requires that signed responses need to include a version number or timestamp, but allows a server to provide a single signature covering all valid versions.

2. A replacement for the whole exchange's signature. This requires the publisher to separately re-sign each valid version and requires each version to include a different update URL, but allows intermediates to serve less data. This is the approach taken in Section 3.

3. A replacement for the exchange's signature and an update for the embedded "expires" and related cache-control HTTP headers [RFC7234]. This naturally extends publishers' intuitions about cache expiration and the existing cache revalidation behavior to signed exchanges. This is sketched and its downsides explored in Appendix C.

The signature also needs to include instructions to intermediates for how to fetch updated validity assertions.

B.4. Low implementation complexity

Simpler implementations are, all things equal, less likely to include bugs. This section describes decisions that were made in the rest of the specification to reduce complexity.

B.4.1. Limited choices

In general, we're trying to eliminate unnecessary choices in the specification. For example, instead of requiring clients to support two methods for verifying payload integrity, we only require one.

B.4.2. Bounded-buffering integrity checking

Clients can be designed with a more-trusted network layer that decides how to trust resources and then provides those resources to less-trusted rendering processes along with handles to the storage and other resources they're allowed to access. If the network layer can enforce that it only operates on chunks of data up to a certain size, it can avoid the complexity of spooling large files to disk.

To allow the network layer to verify signed exchanges using a bounded amount of memory, Section 5.3 requires the signature to be less than 16kB and the headers to be less than 512kB, and Section 3.5 requires that the MI record size be less than 16kB. This allows the network layer to validate a bounded chunk at a time, and pass that chunk on to a renderer, and then forget about that chunk before processing the next one.

The "Digest" header field from [RFC3230] requires the network layer to buffer the entire response body, so it's disallowed.

Appendix C. Determining validity using cache control

This draft could expire signature validity using the normal HTTP cache control headers ([RFC7234]) instead of embedding an expiration date in the signature itself. This section specifies how that would work, and describes why I haven't chosen that option.

The signatures in the "Signature" header field (Section 3.1) would no longer contain "date" or "expires" fields.

The validity-checking algorithm (Section 3.5) would initialize "date" from the resource's "Date" header field (Section 7.1.1.2 of [RFC7231]) and initialize "expires" from either the "Expires" header field (Section 5.3 of [RFC7234]) or the "Cache-Control" header field's "max-age" directive (Section 5.2.2.8 of [RFC7234]) (added to "date"), whichever is present, preferring "max-age" (or failing) if both are present.

Validity updates (Section 3.6) would include a list of replacement response header fields. For each header field name in this list, the client would remove matching header fields from the stored exchange's response header fields. Then the client would append the replacement header fields to the stored exchange's response header fields.

C.1. Example of updating cache control

For example, given a stored exchange of:

```
GET  / HTTP/1.1
Host: example.com
Accept: */*

HTTP/1.1 200
Date: Mon, 20 Nov 2017 10:00:00 UTC
Content-Type: text/html
Date: Tue, 21 Nov 2017 10:00:00 UTC
Expires: Sun, 26 Nov 2017 10:00:00 UTC

<!doctype html>
<html>
...
```

And an update listing the following headers:

```
Expires: Fri, 1 Dec 2017 10:00:00 UTC
Date: Sat, 25 Nov 2017 10:00:00 UTC
```

The resulting stored exchange would be:

```
GET / HTTP/1.1
Host: example.com
Accept: */*

HTTP/1.1 200
Content-Type: text/html
Expires: Fri, 1 Dec 2017 10:00:00 UTC
Date: Sat, 25 Nov 2017 10:00:00 UTC

<!doctype html>
<html>
...
```

C.2. Downsides of updating cache control

In an exchange with multiple signatures, using cache control to expire signatures forces all signatures to initially live for the same period. Worse, the update from one signature's "validity-url" might not match the update for another signature. Clients would need to maintain a current set of headers for each signature, and then decide which set to use when actually parsing the resource itself.

This need to store and reconcile multiple sets of headers for a single signed exchange argues for embedding a signature's lifetime into the signature.

Appendix D. Change Log

RFC EDITOR PLEASE DELETE THIS SECTION.

draft-09

* No change

draft-08

* Improve the privacy considerations.

draft-07

* Provisionally register application/signed-exchange and application/cert-chain+cbor.

draft-06

* Add a security consideration for future-dated OCSP responses and for stolen private keys.

* Define a CAA parameter to opt into certificate issuance.

* Limit certificate lifetimes to 90 days.

* UTF-8 decode the fallback URL.

draft-05

* Define absolute URLs, and limit the schemes each instance can use.

* Fill in TBD size limits.

* Update to mice-03 including the Digest header.

* Refer to draft-yasskin-httpbis-origin-signed-exchanges-impl for draft version numbers.

* Require "exchange"'s response to be cachable by a shared cache.

* Define the "integrity" field of the Signature header to include subfields of the main integrity-protecting header, including the digest algorithm.

* Put a fallback URL at the beginning of the "application/signed-exchange" format, which replaces the ':url' key from the CBOR representation of the exchange's request and response metadata and headers.

* Remove the rest of the request headers from the signed data, in favor of representing content negotiation with the "Variants" response header.

* Make the signed message format a concatenation of byte sequences, which helps implementations avoid re-serializing the exchange's request and response metadata and headers.

* Explicitly check the response payload's integrity instead of assuming the client did it elsewhere in processing the response.

* Reject uncached header fields.

* Update to draft-ietf-httpbis-header-structure-09.

* Update to the final TLS 1.3 RFC.

draft-04

* Update to draft-ietf-httpbis-header-structure-06.

* Replace the application/http-exchange+cbor format with a simpler application/signed-exchange format that:

  - Doesn't require a streaming CBOR parser parse it from a network stream.

  - Doesn't allow request payloads or response trailers, which don't fit into the signature model.

  - Allows checking the signature before parsing the exchange headers.

* Require absolute URLs.

* Make all identifiers in headers lower-case, as required by Structured Headers.

* Switch back to the TLS 1.3 signature format.

* Include the version and draft number in the signature context string.

* Remove support for integrity protection using the Digest header field.

* Limit the record size in the mi-sha256 encoding.

* Forbid RSA keys, and only require clients to support secp256r1 keys.

* Add a test OID for the CanSignHttpExchanges X.509 extension.

draft-03

* Allow each method of transferring an exchange to define which headers are signed, have the cross-origin methods use all headers, and remove the "allResponseHeaders" flag.

* Describe footguns around signing private content, and block certain headers to make it less likely.

* Define a CBOR structure to hold the certificate chain instead of re-using the TLS1.3 message. The TLS 1.3 parser fails on unexpected extensions while this format should ignore them, and apparently TLS implementations don't expose their message parsers enough to allow passing a message to a certificate verifier.

* Require an X.509 extension for the signing certificate.

draft-02

* Signatures identify a header (e.g. Digest or MI) to guard the payload's integrity instead of directly signing over the payload.

* The validityUrl is signed.

* Use CBOR maps where appropriate, and define how they're canonicalized.

* Remove the update.url field from signature validity updates, in favor of just re-fetching the original request URL.

* Define an HTTP/2 extension to use a setting to enable cross-origin Server Push.

* Define an "Accept-Signature" header to negotiate whether to send Signatures and which ones.

* Define an "application/http-exchange+cbor" format to fetch signed exchanges without HTTP/2 Push.

* 2 new use cases.

Appendix E. Acknowledgements

Thanks to Andrew Ayer, Devin Mullins, Ilari Liusvaara, John Wilander, Justin Schuh, Mark Nottingham, Mike Bishop, Ryan Sleevi, and Yoav Weiss for comments that improved this draft.

Author's Address

Jeffrey Yasskin
Google

Email: jyasskin@chromium.org