

HTTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2016

M. Nottingham
Akamai
P. McManus
Mozilla
J. Reschke
greenbytes
March 8, 2016

HTTP Alternative Services
draft-ietf-httpbis-alt-svc-14

Abstract

This document specifies "Alternative Services" for HTTP, which allow an origin's resources to be authoritatively available at a separate network location, possibly accessed with a different protocol configuration.

Editorial Note (To be removed by RFC Editor)

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

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

The changes in this draft are summarized in Appendix A.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 9, 2016.

Copyright Notice

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

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

Table of Contents

1. Introduction	4
1.1. Notational Conventions	4
2. Alternative Services Concepts	5
2.1. Host Authentication	7
2.2. Alternative Service Caching	7
2.3. Requiring Server Name Indication	8
2.4. Using Alternative Services	8
3. The Alt-Svc HTTP Header Field	9
3.1. Caching Alt-Svc Header Field Values	11
4. The ALTSVC HTTP/2 Frame	12
5. The Alt-Used HTTP Header Field	14
6. The 421 Misdirected Request HTTP Status Code	14
7. IANA Considerations	15
7.1. Header Field Registrations	15
7.2. The ALTSVC HTTP/2 Frame Type	15
7.3. Alt-Svc Parameter Registry	15
7.3.1. Procedure	15
7.3.2. Registrations	16
8. Internationalization Considerations	16
9. Security Considerations	16
9.1. Changing Ports	16
9.2. Changing Hosts	17
9.3. Changing Protocols	17
9.4. Tracking Clients Using Alternative Services	18
9.5. Confusion Regarding Request Scheme	18
10. References	19
10.1. Normative References	19
10.2. Informative References	20
Appendix A. Change Log (to be removed by RFC Editor before publication)	20
A.1. Since draft-nottingham-httpbis-alt-svc-05	20
A.2. Since draft-ietf-httpbis-alt-svc-00	21
A.3. Since draft-ietf-httpbis-alt-svc-01	21
A.4. Since draft-ietf-httpbis-alt-svc-02	21
A.5. Since draft-ietf-httpbis-alt-svc-03	21
A.6. Since draft-ietf-httpbis-alt-svc-04	21
A.7. Since draft-ietf-httpbis-alt-svc-05	22
A.8. Since draft-ietf-httpbis-alt-svc-06	22
A.9. Since draft-ietf-httpbis-alt-svc-07	22
A.10. Since draft-ietf-httpbis-alt-svc-08	23
A.11. Since draft-ietf-httpbis-alt-svc-09	24
A.12. Since draft-ietf-httpbis-alt-svc-10	24
A.13. Since draft-ietf-httpbis-alt-svc-11	24
A.14. Since draft-ietf-httpbis-alt-svc-12	24
Appendix B. Acknowledgements	24

1. Introduction

HTTP [RFC7230] conflates the identification of resources with their location. In other words, "http://" and "https://" URIs are used to both name and find things to interact with.

In some cases, it is desirable to separate identification and location in HTTP; keeping the same identifier for a resource, but interacting with it at a different location on the network.

For example:

- o An origin server might wish to redirect a client to a different server when it is under load, or it has found a server in a location that is more local to the client.
- o An origin server might wish to offer access to its resources using a new protocol, such as HTTP/2 [RFC7540], or one using improved security, such as Transport Layer Security (TLS) [RFC5246].
- o An origin server might wish to segment its clients into groups of capabilities, such as those supporting Server Name Indication (SNI) (Section 3 of [RFC6066]), for operational purposes.

This specification defines a new concept in HTTP, "Alternative Services", that allows an origin server to nominate additional means of interacting with it on the network. It defines a general framework for this in Section 2, along with specific mechanisms for advertising their existence using HTTP header fields (Section 3) or HTTP/2 frames (Section 4), plus a way to indicate that an alternative service was used (Section 5).

It also endorses the status code 421 (Misdirected Request) (Section 6) that origin servers or their nominated alternatives can use to indicate that they are not authoritative for a given origin, in cases where the wrong location is used.

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].

This document uses the Augmented BNF defined in [RFC5234] and updated by [RFC7405] along with the "#rule" extension defined in Section 7 of [RFC7230]. The rules below are defined in [RFC5234], [RFC7230], and [RFC7234]:

OWS = <OWS, see [RFC7230], Section 3.2.3>
delta-seconds = <delta-seconds; see [RFC7234], Section 1.2.1>
port = <port, see [RFC7230], Section 2.7>
quoted-string = <quoted-string, see [RFC7230], Section 3.2.6>
token = <token, see [RFC7230], Section 3.2.6>
uri-host = <uri-host, see [RFC7230], Section 2.7>

2. Alternative Services Concepts

This specification defines a new concept in HTTP, the "Alternative Service". When an origin [RFC6454] has resources that are accessible through a different protocol / host / port combination, it is said to have an alternative service available.

An alternative service can be used to interact with the resources on an origin server at a separate location on the network, possibly using a different protocol configuration. Alternative services are considered authoritative for an origin's resources, in the sense of [RFC7230], Section 9.1.

For example, an origin:

```
("http", "www.example.com", "80")
```

might declare that its resources are also accessible at the alternative service:

```
("h2", "new.example.com", "81")
```

By their nature, alternative services are explicitly at the granularity of an origin; they cannot be selectively applied to resources within an origin.

Alternative services do not replace or change the origin for any given resource; in general, they are not visible to the software "above" the access mechanism. The alternative service is essentially alternative routing information that can also be used to reach the origin in the same way that DNS CNAME or SRV records define routing information at the name resolution level. Each origin maps to a set of these routes -- the default route is derived from the origin itself and the other routes are introduced based on alternative-service information.

Furthermore, it is important to note that the first member of an alternative service tuple is different from the "scheme" component of an origin; it is more specific, identifying not only the major version of the protocol being used, but potentially communication options for that protocol.

This means that clients using an alternative service can change the host, port and protocol that they are using to fetch resources, but these changes MUST NOT be propagated to the application that is using HTTP; from that standpoint, the URI being accessed and all information derived from it (scheme, host, port) are the same as before.

Importantly, this includes its security context; in particular, when TLS [RFC5246] is used to authenticate, the alternative service will need to present a certificate for the origin's host name, not that of the alternative. Likewise, the Host header field ([RFC7230], Section 5.4) is still derived from the origin, not the alternative service (just as it would if a CNAME were being used).

The changes MAY, however, be made visible in debugging tools, consoles, etc.

Formally, an alternative service is identified by the combination of:

- o An Application Layer Protocol Negotiation (ALPN) protocol name, as per [RFC7301]
- o A host, as per [RFC3986], Section 3.2.2
- o A port, as per [RFC3986], Section 3.2.3

The ALPN protocol name is used to identify the application protocol or suite of protocols used by the alternative service. Note that for the purpose of this specification, an ALPN protocol name implicitly includes TLS in the suite of protocols it identifies, unless specified otherwise in its definition. In particular, the ALPN name "http/1.1", registered by Section 6 of [RFC7301], identifies HTTP/1.1 over TLS.

Additionally, each alternative service MUST have:

- o A freshness lifetime, expressed in seconds; see Section 2.2

There are many ways that a client could discover the alternative service(s) associated with an origin. This document describes two such mechanisms: the "Alt-Svc" HTTP header field (Section 3) and the "ALTSVC" HTTP/2 frame type (Section 4).

The remainder of this section describes requirements that are common to alternative services, regardless of how they are discovered.

2.1. Host Authentication

Clients **MUST** have reasonable assurances that the alternative service is under control of and valid for the whole origin. This mitigates the attack described in Section 9.2.

For the purposes of this document, "reasonable assurances" can be established through use of a TLS-based protocol with the certificate checks defined in [RFC2818]. Clients **MAY** impose additional criteria for establishing reasonable assurances.

For example, if the origin's host is "www.example.com" and an alternative is offered on "other.example.com" with the "h2" protocol, and the certificate offered is valid for "www.example.com", the client can use the alternative. However, if either is offered with the "h2c" protocol, the client cannot use it, because there is no mechanism (at the time of the publication of this specification) in that protocol to establish the relationship between the origin and the alternative.

2.2. Alternative Service Caching

Mechanisms for discovering alternative services also associate a freshness lifetime with them; for example, the Alt-Svc header field uses the "ma" parameter.

Clients can choose to use an alternative service instead of the origin at any time when it is considered fresh; see Section 2.4 for specific recommendations.

Clients with existing connections to an alternative service do not need to stop using it when its freshness lifetime ends; the caching mechanism is intended for limiting how long an alternative service can be used for establishing new connections, not limiting the use of existing ones.

Alternative services are fully authoritative for the origin in question, including the ability to clear or update cached alternative service entries, extend freshness lifetimes, and any other authority the origin server would have.

When alternative services are used to send a client to the most optimal server, a change in network configuration can result in cached values becoming suboptimal. Therefore, clients **SHOULD** remove from cache all alternative services that lack the "persist" flag with the value "1" when they detect such a change, when information about network state is available.

2.3. Requiring Server Name Indication

A client **MUST NOT** use a TLS-based alternative service unless the client supports TLS Server Name Indication (SNI). This supports the conservation of IP addresses on the alternative service host.

Note that the SNI information provided in TLS by the client will be that of the origin, not the alternative (as will the Host HTTP header field value).

2.4. Using Alternative Services

By their nature, alternative services are **OPTIONAL**: clients do not need to use them. However, it is advantageous for clients to behave in a predictable way when alternative services are used by servers, to aid purposes like load balancing.

Therefore, if a client supporting this specification becomes aware of an alternative service, the client **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. A viable alternative service is then treated in every way as the origin; this includes the ability to advertise alternative services.

If a client becomes aware of multiple alternative services, it chooses the most suitable according to its own criteria, keeping security properties in mind. For example, an origin might advertise multiple alternative services to notify clients of support for multiple versions of HTTP.

A client configured to use a proxy for a given request **SHOULD NOT** directly connect to an alternative service for this request, but instead route it through that proxy.

When a client uses an alternative service for a request, it can indicate this to the server using the Alt-Used header field (Section 5).

The client does not need to block requests on any existing connection; it can be used until the alternative connection is established. However, if the security properties of the existing connection are weak (for example, cleartext HTTP/1.1) then it might make sense to block until the new connection is fully available in order to avoid information leakage.

Furthermore, if the connection to the alternative service fails or is

unresponsive, the client MAY fall back to using the origin or another alternative service. Note, however, that this could be the basis of a downgrade attack, thus losing any enhanced security properties of the alternative service. If the connection to the alternative service does not negotiate the expected protocol (for example, ALPN fails to negotiate h2, or an Upgrade request to h2c is not accepted), the connection to the alternative service MUST be considered to have failed.

3. The Alt-Svc HTTP Header Field

An HTTP(S) origin server can advertise the availability of alternative services to clients by adding an Alt-Svc header field to responses.

```
Alt-Svc      = clear / 1#alt-value
clear        = %s"clear"; "clear", case-sensitive
alt-value    = alternative *( OWS ";" OWS parameter )
alternative  = protocol-id "=" alt-authority
protocol-id  = token ; percent-encoded ALPN protocol name
alt-authority = quoted-string ; containing [ uri-host ] ":" port
parameter    = token "=" ( token / quoted-string )
```

The field value consists either of a list of values, each of which indicates one alternative service, or the keyword "clear".

A field value containing the special value "clear" indicates that the origin requests all alternatives for that origin to be invalidated (including those specified in the same response, in case of an invalid reply containing both "clear" and alternative services).

ALPN protocol names are octet sequences with no additional constraints on format. Octets not allowed in tokens ([RFC7230], Section 3.2.6) MUST be percent-encoded as per Section 2.1 of [RFC3986]. Consequently, the octet representing the percent character "%" (hex 25) MUST be percent-encoded as well.

In order to have precisely one way to represent any ALPN protocol name, the following additional constraints apply:

1. Octets in the ALPN protocol name MUST NOT be percent-encoded if they are valid token characters except "%", and
2. When using percent-encoding, uppercase hex digits MUST be used.

With these constraints, recipients can apply simple string comparison to match protocol identifiers.

The "alt-authority" component consists of an OPTIONAL uri-host ("host" in Section 3.2.2 of [RFC3986]), a colon (":"), and a port number.

For example:

```
Alt-Svc: h2=":8000"
```

This indicates the "h2" protocol ([RFC7540]) on the same host using the indicated port 8000.

An example involving a change of host:

```
Alt-Svc: h2="new.example.org:80"
```

This indicates the "h2" protocol on the host "new.example.org", running on port 80. Note that the "quoted-string" syntax needs to be used because ":" is not an allowed character in "token".

Examples for protocol name escaping:

ALPN protocol name	protocol-id	Note
h2	h2	No escaping needed
w=x:y#z	w%3Dx%3Ay#z	"=" and ":" escaped
x%y	x%25y	"%" needs escaping

Alt-Svc MAY occur in any HTTP response message, regardless of the status code. Note that recipients of Alt-Svc can ignore the header field (and are required to in some situations; see Sections 2.1 and 6).

The Alt-Svc field value can have multiple values:

```
Alt-Svc: h2="alt.example.com:8000", h2=":443"
```

When multiple values are present, the order of the values reflects the server's preference (with the first value being the most preferred alternative).

The value(s) advertised by Alt-Svc can be used by clients to open a new connection to an alternative service. Subsequent requests can start using this new connection immediately, or can continue using the existing connection while the new connection is created.

When using HTTP/2 ([RFC7540]), servers SHOULD instead send an ALTSVC frame (Section 4). A single ALTSVC frame can be sent for a connection; a new frame is not needed for every request. Note that, despite this recommendation, Alt-Svc header fields remain valid in responses delivered over HTTP/2.

Each "alt-value" is followed by an OPTIONAL semicolon-separated list of additional parameters, each such "parameter" comprising a name and a value.

This specification defines two parameters: "ma" and "persist", defined in Section 3.1. Unknown parameters MUST be ignored. That is, the values (alt-value) they appear in MUST be processed as if the unknown parameter was not present.

New parameters can be defined in extension specifications (see Section 7.3 for registration details).

Note that all field elements that allow "quoted-string" syntax MUST be processed as per Section 3.2.6 of [RFC7230].

3.1. Caching Alt-Svc Header Field Values

When an alternative service is advertised using Alt-Svc, it is considered fresh for 24 hours from generation of the message. This can be modified with the 'ma' (max-age) parameter.

Syntax:

ma = delta-seconds; see [RFC7234], Section 1.2.1

The delta-seconds value indicates the number of seconds since the response was generated the alternative service is considered fresh for.

Alt-Svc: h2=":443"; ma=3600

See Section 4.2.3 of [RFC7234] for details of determining response age.

For example, a response:

```
HTTP/1.1 200 OK
Content-Type: text/html
Cache-Control: max-age=600
Age: 30
Alt-Svc: h2=":8000"; ma=60
```

indicates that an alternative service is available and usable for the next 60 seconds. However, the response has already been cached for 30 seconds (as per the Age header field value), so therefore the alternative service is only fresh for the 30 seconds from when this response was received, minus estimated transit time.

Note that the freshness lifetime for HTTP caching (here, 600 seconds) does not affect caching of Alt-Svc values.

When an Alt-Svc response header field is received from an origin, its value invalidates and replaces all cached alternative services for that origin.

By default, cached alternative services will be cleared when the client detects a network change. Alternative services that are intended to be longer-lived (such as those that are not specific to the client access network) can carry the "persist" parameter with a value "1" as a hint that the service is potentially useful beyond a network configuration change.

Syntax:

```
persist = "1"
```

For example:

```
Alt-Svc: h2=":443"; ma=2592000; persist=1
```

This specification only defines a single value for "persist". Clients MUST ignore "persist" parameters with values other than "1".

See Section 2.2 for general requirements on caching alternative services.

4. The ALTSVC HTTP/2 Frame

The ALTSVC HTTP/2 frame ([RFC7540], Section 4) advertises the availability of an alternative service to an HTTP/2 client.

The ALTSVC frame is a non-critical extension to HTTP/2. Endpoints

that do not support this frame will ignore it (as per the extensibility rules defined in Section 4.1 of [RFC7540]).

An ALTSVC frame from a server to a client on a stream other than stream 0 indicates that the conveyed alternative service is associated with the origin of that stream.

An ALTSVC frame from a server to a client on stream 0 indicates that the conveyed alternative service is associated with the origin contained in the Origin field of the frame. An association with an origin that the client does not consider authoritative for the current connection **MUST** be ignored.

The ALTSVC frame type is 0xa (decimal 10).

Origin-Len (16)	Origin? (*)	...
Alt-Svc-Field-Value (*)		...

ALTSVC Frame Payload

The ALTSVC frame contains the following fields:

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

Origin: An OPTIONAL sequence of characters containing the ASCII serialization of an origin ([RFC6454], Section 6.2) that the alternative service is applicable to.

Alt-Svc-Field-Value: A sequence of octets (length determined by subtracting the length of all preceding fields from the frame length) containing a value identical to the Alt-Svc field value defined in Section 3 (ABNF production "Alt-Svc").

The ALTSVC frame does not define any flags.

The ALTSVC frame is intended for receipt by clients. A device acting as a server **MUST** ignore it.

An ALTSVC frame on stream 0 with empty (length 0) "Origin" information is invalid and **MUST** be ignored. An ALTSVC frame on a stream other than stream 0 containing non-empty "Origin" information is invalid and **MUST** be ignored.

The ALTSVC frame is processed hop-by-hop. An intermediary **MUST NOT**

forward ALTSVC frames, though it can use the information contained in ALTSVC frames in forming new ALTSVC frames to send to its own clients.

Receiving an ALTSVC frame is semantically equivalent to receiving an Alt-Svc header field. As a result, the ALTSVC frame causes alternative services for the corresponding origin to be replaced. Note that it would be unwise to mix the use of Alt-Svc header fields with the use of ALTSVC frames, as the sequence of receipt might be hard to predict.

5. The Alt-Used HTTP Header Field

The Alt-Used header field is used in requests to indicate the identity of the alternative service in use, just as the Host header field (Section 5.4 of [RFC7230]) identifies the host and port of the origin.

Alt-Used = uri-host [":" port]

Alt-Used is intended to allow alternative services to detect loops, differentiate traffic for purposes of load balancing, and generally to ensure that it is possible to identify the intended destination of traffic, since introducing this information after a protocol is in use has proven to be problematic.

When using an alternative service, clients SHOULD include an Alt-Used header field in all requests.

For example:

```
GET /thing HTTP/1.1
Host: origin.example.com
Alt-Used: alternate.example.net
```

6. The 421 Misdirected Request HTTP Status Code

The 421 (Misdirected Request) status code is defined in Section 9.1.2 of [RFC7540] to indicate that the current server instance is not authoritative for the requested resource. This can be used to indicate that an alternative service is not authoritative; see Section 2).

Clients receiving 421 (Misdirected Request) from an alternative service MUST remove the corresponding entry from its alternative service cache (see Section 2.2) for that origin. Regardless of the idempotency of the request method, they MAY retry the request, either at another alternative server, or at the origin.

An Alt-Svc header field in a 421 (Misdirected Request) response MUST be ignored.

7. IANA Considerations

7.1. Header Field Registrations

HTTP header fields are registered within the "Message Headers" registry maintained at <https://www.iana.org/assignments/message-headers/>.

This document defines the following HTTP header fields, so their associated registry entries shall be added according to the permanent registrations below (see [BCP90]):

Header Field Name	Protocol	Status	Reference
Alt-Svc	http	standard	Section 3
Alt-Used	http	standard	Section 5

The change controller is: "IETF (iesg@ietf.org) - Internet Engineering Task Force".

7.2. The ALTSVC HTTP/2 Frame Type

This document registers the ALTSVC frame type in the HTTP/2 Frame Types registry ([RFC7540], Section 11.2).

Frame Type: ALTSVC

Code: 0xa

Specification: Section 4 of this document

7.3. Alt-Svc Parameter Registry

The HTTP Alt-Svc Parameter Registry defines the name space for parameters. It will be created and maintained at (the suggested URI) <http://www.iana.org/assignments/http-alt-svc-parameters>.

7.3.1. Procedure

A registration MUST include the following fields:

- o Parameter Name

- o Pointer to specification text

Values to be added to this name space require Expert Review (see [RFC5226], Section 4.1).

7.3.2. Registrations

The HTTP Alt-Svc Parameter Registry is to be populated with the registrations below:

Alt-Svc Parameter	Reference
ma	Section 3.1
persist	Section 3.1

8. Internationalization Considerations

An internationalized domain name that appears in either the header field (Section 3) or the HTTP/2 frame (Section 4) MUST be expressed using A-labels ([RFC5890], Section 2.3.2.1).

9. Security Considerations

9.1. Changing Ports

Using an alternative service implies accessing an origin's resources on an alternative port, at a minimum. An attacker that can inject alternative services and listen at the advertised port is therefore able to hijack an origin. On certain servers, it is normal for users to be able to control some personal pages available on a shared port, and also to accept to requests on less-privileged ports.

For example, an attacker that can add HTTP response header fields to some pages can redirect traffic for an entire origin to a different port on the same host using the Alt-Svc header field; if that port is under the attacker's control, they can thus masquerade as the HTTP server.

This risk is mitigated by the requirements in Section 2.1.

On servers, this risk can also be reduced by restricting the ability to advertise alternative services, and restricting who can open a port for listening on that host.

9.2. Changing Hosts

When the host is changed due to the use of an alternative service, it presents an opportunity for attackers to hijack communication to an origin.

For example, if an attacker can convince a user agent to send all traffic for "innocent.example.org" to "evil.example.com" by successfully associating it as an alternative service, they can masquerade as that origin. This can be done locally (see mitigations in Section 9.1) or remotely (e.g., by an intermediary as a man-in-the-middle attack).

This is the reason for the requirement in Section 2.1 that clients have reasonable assurances that the alternative service is under control of and valid for the whole origin; for example, presenting a certificate for the origin proves that the alternative service is authorized to serve traffic for the origin.

Note that this assurance is only as strong as the method used to authenticate the alternative service. In particular, when TLS authentication is used to do so, there are well-known exploits to make an attacker's certificate appear as legitimate.

Alternative services could be used to persist such an attack. For example, an intermediary could man-in-the-middle TLS-protected communication to a target, and then direct all traffic to an alternative service with a large freshness lifetime, so that the user agent still directs traffic to the attacker even when not using the intermediary.

Implementations MUST perform any certificate-pinning validation (such as [RFC7469]) on alternative services just as they would on direct connections to the origin. Implementations might also choose to add other requirements around which certificates are acceptable for alternative services.

9.3. Changing Protocols

When the ALPN protocol is changed due to the use of an alternative service, the security properties of the new connection to the origin can be different from that of the "normal" connection to the origin, because the protocol identifier itself implies this.

For example, if an "https://" URI has a protocol advertised that does not use some form of end-to-end encryption (most likely, TLS), it violates the expectations for security that the URI scheme implies. Therefore, clients cannot blindly use alternative services, but

instead evaluate the option(s) presented to assure that security requirements and expectations of specifications, implementations and end users are met.

9.4. Tracking Clients Using Alternative Services

Choosing an alternative service implies connecting to a new, server-supplied host name. By using unique names, servers could conceivably track client requests. Such tracking could follow users across multiple networks, when the "persist" flag is used.

Clients that wish to prevent requests from being correlated can decide not to use alternative services for multiple requests that would not otherwise be allowed to be correlated.

In a user agent, any alternative service information **MUST** be removed when origin-specific data is cleared (typically, when cookies [RFC6265] are cleared).

9.5. Confusion Regarding Request Scheme

Some server-side HTTP applications make assumptions about security based upon connection context; for example, equating being served upon port 443 with the use of an "https://" URI and the various security properties that implies.

This affects not only the security properties of the connection itself, but also the state of the client at the other end of it; for example, a Web browser treats "https://" URIs differently than "http://" URIs in many ways, not just for purposes of protocol handling.

Since one of the uses of Alternative Services is to allow a connection to be migrated to a different protocol and port, these applications can become confused about the security properties of a given connection, sending information (for example, cookies and content) that is intended for a secure context (such as an "https://" URI) to a client that is not treating it as one.

This risk can be mitigated in servers by using the URI scheme explicitly carried by the protocol (such as ":scheme" in HTTP/2 or the "absolute form" of the request target in HTTP/1.1) as an indication of security context, instead of other connection properties ([RFC7540], Section 8.1.2.3 and [RFC7230], Section 5.3.2).

When the protocol does not explicitly carry the scheme (as is usually the case for HTTP/1.1 over TLS), servers can mitigate this risk by either assuming that all requests have an insecure context, or by

refraining from advertising alternative services for insecure schemes (for example, HTTP).

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<http://www.rfc-editor.org/info/rfc2818>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, DOI 10.17487/RFC5890, August 2010, <<http://www.rfc-editor.org/info/rfc5890>>.
- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<http://www.rfc-editor.org/info/rfc6066>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<http://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, <<http://www.rfc-editor.org/info/rfc7230>>.

- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<http://www.rfc-editor.org/info/rfc7234>>.
- [RFC7301] Friedl, S., Popov, A., Langley, A., and S. Emile, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<http://www.rfc-editor.org/info/rfc7301>>.
- [RFC7405] Kyzivat, P., "Case-Sensitive String Support in ABNF", RFC 7405, DOI 10.17487/RFC7405, December 2014, <<http://www.rfc-editor.org/info/rfc7405>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol version 2", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<http://www.rfc-editor.org/info/rfc7540>>.

10.2. Informative References

- [BCP90] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004, <<http://www.rfc-editor.org/info/bcp90>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<http://www.rfc-editor.org/info/rfc6265>>.
- [RFC7469] Evans, C., Palmer, C., and R. Sleevi, "Public Key Pinning Extension for HTTP", RFC 7469, DOI 10.17487/RFC7469, April 2015, <<http://www.rfc-editor.org/info/rfc7469>>.

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

A.1. Since draft-nottingham-httpbis-alt-svc-05

This is the first version after adoption of draft-nottingham-httpbis-alt-svc-05 as Working Group work item. It only contains editorial changes.

A.2. Since draft-ietf-httpbis-alt-svc-00

Selected 421 as proposed status code for "Not Authoritative".

Changed header field syntax to use percent-encoding of ALPN protocol names (<<https://github.com/http2/http2-spec/issues/446>>).

A.3. Since draft-ietf-httpbis-alt-svc-01

Updated HTTP/1.1 references.

Renamed "Service" to "Alt-Svc-Used" and reduced information to a flag to address fingerprinting concerns (<<https://github.com/http2/http2-spec/issues/502>>).

Note that ALTSVC frame is preferred to Alt-Svc header field (<<https://github.com/http2/http2-spec/pull/503>>).

Incorporate ALTSRV frame (<<https://github.com/http2/http2-spec/pull/507>>).

Moved definition of status code 421 to HTTP/2.

Partly resolved <<https://github.com/httpwg/http-extensions/issues/5>>.

A.4. Since draft-ietf-httpbis-alt-svc-02

Updated ALPN reference.

Resolved <<https://github.com/httpwg/http-extensions/issues/2>>.

A.5. Since draft-ietf-httpbis-alt-svc-03

Renamed "Alt-Svc-Used" to "Alt-Used" (<<https://github.com/httpwg/http-extensions/issues/17>>).

Clarify ALTSVC Origin information requirements (<<https://github.com/httpwg/http-extensions/issues/19>>).

Remove/tune language with respect to tracking risks (see <<https://github.com/httpwg/http-extensions/issues/34>>).

A.6. Since draft-ietf-httpbis-alt-svc-04

Mention tracking by alt-svc host name in Security Considerations (<<https://github.com/httpwg/http-extensions/issues/36>>).

"421 (Not Authoritative)" -> "421 (Misdirected Request)".

Allow the frame to carry multiple indicator and use the same payload formats for both
(<https://github.com/httpwg/http-extensions/issues/37>).

A.7. Since draft-ietf-httpbis-alt-svc-05

Go back to specifying the origin in Alt-Used, but make it a "SHOULD"
(<https://github.com/httpwg/http-extensions/issues/34>).

Restore Origin field in ALT-SVC frame
(<https://github.com/httpwg/http-extensions/issues/38>).

A.8. Since draft-ietf-httpbis-alt-svc-06

Disallow use of alternative services when the protocol might not carry the scheme
(<https://github.com/httpwg/http-extensions/issues/12>).

Align opp-sec and alt-svc
(<https://github.com/httpwg/http-extensions/issues/33>).

alt svc frame on pushed (even and non-0) frame
(<https://github.com/httpwg/http-extensions/issues/44>).

"browser" -> "user agent"
(<https://github.com/httpwg/http-extensions/pull/61>).

ABNF for "parameter"
(<https://github.com/httpwg/http-extensions/issues/65>).

Updated HTTP/2 reference.

A.9. Since draft-ietf-httpbis-alt-svc-07

Alt-Svc alternative cache invalidation
(<https://github.com/httpwg/http-extensions/issues/16>).

Unexpected Alt-Svc frames
(<https://github.com/httpwg/http-extensions/issues/18>).

Associating Alt-Svc header with an origin
(<https://github.com/httpwg/http-extensions/issues/21>).

ALPN identifiers in Alt-Svc
(<https://github.com/httpwg/http-extensions/issues/43>).

Number of alternate services used
(<https://github.com/httpwg/http-extensions/issues/58>).

Proxy and .pac interaction

(<https://github.com/httpwg/http-extensions/issues/62>).

Need to define extensibility for alt-svc parameters

(<https://github.com/httpwg/http-extensions/issues/69>).

Persistence of alternates across network changes

(<https://github.com/httpwg/http-extensions/issues/71>).

Alt-Svc header with 421 status

(<https://github.com/httpwg/http-extensions/issues/75>).

Incorporate several editorial improvements suggested by Mike Bishop

(<https://github.com/httpwg/http-extensions/pull/77>,

<https://github.com/httpwg/http-extensions/pull/78>).

Alt-Svc response header field in HTTP/2 frame

(<https://github.com/httpwg/http-extensions/issues/87>).

A.10. Since draft-ietf-httpbis-alt-svc-08

Remove left over text about ext-params, applying to an earlier version of Alt-Used (see

<https://github.com/httpwg/http-extensions/issues/34>).

Conflicts between Alt-Svc and ALPN

(<https://github.com/httpwg/http-extensions/issues/72>).

Elevation of privilege

(<https://github.com/httpwg/http-extensions/issues/73>).

Alternates of alternates

(<https://github.com/httpwg/http-extensions/issues/74>).

Alt-Svc and Cert Pinning

(<https://github.com/httpwg/http-extensions/issues/76>).

Using alt-svc on localhost (no change to spec, see

<https://github.com/httpwg/http-extensions/issues/89>).

IANA procedure for alt-svc parameters

(<https://github.com/httpwg/http-extensions/issues/96>).

Alt-svc from https (1.1) to https (1.1)

(<https://github.com/httpwg/http-extensions/issues/91>).

Alt-svc vs the ability to convey the scheme inside the protocol

(<https://github.com/httpwg/http-extensions/issues/92>).

Reconciling MAY/can vs. SHOULD
(<https://github.com/httpwg/http-extensions/issues/101>).

Typo in alt-svc caching example
(<https://github.com/httpwg/http-extensions/issues/117>).

A.11. Since draft-ietf-httpbis-alt-svc-09

Editorial improvements
(<https://github.com/httpwg/http-extensions/issues/118>,
<https://github.com/httpwg/http-extensions/issues/119>,
<https://github.com/httpwg/http-extensions/issues/120>,
<https://github.com/httpwg/http-extensions/issues/121>,
<https://github.com/httpwg/http-extensions/issues/122>,
<https://github.com/httpwg/http-extensions/issues/123>,
<https://github.com/httpwg/http-extensions/issues/125>,
<https://github.com/httpwg/http-extensions/issues/126>).

A.12. Since draft-ietf-httpbis-alt-svc-10

Editorial improvements
(<https://github.com/httpwg/http-extensions/issues/130>).

Use RFC 7405 ABNF extension
(<https://github.com/httpwg/http-extensions/issues/131>).

A.13. Since draft-ietf-httpbis-alt-svc-11

Security considerations wrt system ports
(<https://github.com/httpwg/http-extensions/issues/139>).

A.14. Since draft-ietf-httpbis-alt-svc-12

Editorial changes triggered by <https://lists.w3.org/Archives/Public/ietf-http-wg/2016JanMar/0243.html>.

Reasonable Assurances and H2C
(<https://github.com/httpwg/http-extensions/issues/148>).

Appendix B. Acknowledgements

Thanks to Adam Langley, Bence Beky, Chris Lonvick, Eliot Lear, Erik Nygren, Guy Podjarny, Herve Ruellan, Lucas Pardue, Martin Thomson, Matthew Kerwin, Mike Bishop, Paul Hoffman, Richard Barnes, Richard Bradbury, Stephen Farrell, Stephen Ludin, and Will Chan for their feedback and suggestions.

The Alt-Svc header field was influenced by the design of the

Alternate-Protocol header field in SPDY.

Authors' Addresses

Mark Nottingham
Akamai

EMail: mnot@mnot.net
URI: <https://www.mnot.net/>

Patrick McManus
Mozilla

EMail: mcmanus@ducksong.com
URI: <https://mozillians.org/u/pmcmanus/>

Julian F. Reschke
greenbytes GmbH

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

HTTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 11, 2016

J. Reschke
greenbytes
September 8, 2015

Hypertext Transfer Protocol (HTTP) Client-Initiated Content-Encoding
draft-ietf-httpbis-cice-03

Abstract

In HTTP, content codings allow for payload encodings such as for compression or integrity checks. In particular, the "gzip" content coding is widely used for payload data sent in response messages.

Content codings can be used in request messages as well, however discoverability is not on par with response messages. This document extends the HTTP "Accept-Encoding" header field for use in responses, to indicate the content codings that are supported in requests.

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 <https://tools.ietf.org/wg/httpbis/> and <http://httpwg.github.io/>; source code and issues list for this draft can be found at <https://github.com/httpwg/http-extensions>.

The changes in this draft are summarized in Appendix A.6.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on March 11, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
2. Notational Conventions	3
3. Using the 'Accept-Encoding' Header Field in Responses	3
4. Example	4
5. Deployment Considerations	5
6. Security Considerations	5
7. IANA Considerations	5
7.1. Header Field Registry	5
7.2. Status Code Registry	6
8. References	6
8.1. Normative References	6
8.2. Informative References	7
Appendix A. Change Log (to be removed by RFC Editor before publication)	7
A.1. Since draft-reschke-http-cice-00	7
A.2. Since draft-reschke-http-cice-01	7
A.3. Since draft-reschke-http-cice-02	7
A.4. Since draft-ietf-httpbis-cice-00	7
A.5. Since draft-ietf-httpbis-cice-01	8
A.6. Since draft-ietf-httpbis-cice-02	8
Appendix B. Acknowledgements	8

1. Introduction

In HTTP, content codings allow for payload encodings such as for compression or integrity checks ([RFC7231], Section 3.1.2). In particular, the "gzip" content coding ([RFC7230], Section 4.2) is widely used for payload data sent in response messages.

Content codings can be used in request messages as well, however discoverability is not on par with response messages. This document extends the HTTP "Accept-Encoding" header field ([RFC7231], Section 5.3.4) for use in responses, to indicate the content codings that are supported in requests. It furthermore updates the definition of status code 415 (Unsupported Media Type) ([RFC7231], Section 6.5.13), recommending to include the "Accept-Encoding" header field when appropriate.

2. Notational Conventions

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

This document reuses terminology defined in the base HTTP specifications, namely Section 2 of [RFC7230] and Section 3.1.2 of [RFC7231].

3. Using the 'Accept-Encoding' Header Field in Responses

Section 5.3.4 of [RFC7231] defines "Accept-Encoding" as a request header field only.

This specification expands that definition to allow "Accept-Encoding" as a response header field as well. When present in a response, it indicates what content codings the resource was willing to accept in the associated request. A field value that only contains "identity" implies that no content codings were supported.

Note that this information is specific to the associated request; the set of supported encodings might be different for other resources on the same server, and could change over time or depend on other aspects of the request (such as the request method).

Section 6.5.13 of [RFC7231] defines status code 415 (Unsupported Media Type) to apply to both media type and content coding related problems.

Servers that fail a request due to an unsupported content coding ought to respond with a 415 status and ought to include an "Accept-

Encoding" header field in that response, allowing clients to distinguish between content coding related issues and media type related issues. In order to avoid confusion with media type related problems, servers that fail a request with a 415 status for reasons unrelated to content codings MUST NOT include the "Accept-Encoding" header field.

It is expected that the most common use of "Accept-Encoding" in responses will have the 415 (Unsupported Media Type) status code, in response to optimistic use of a content coding by clients. However, the header field can also be used to indicate to clients that content codings are supported, to optimize future interactions. For example, a resource might include it in a 2xx response when the request payload was big enough to justify use of a compression coding, but the client failed to do so.

4. Example

A client submits a POST request using the "compress" content coding ([RFC7231], Section 3.1.2.1):

```
POST /edit/ HTTP/1.1
Host: example.org
Content-Type: application/atom+xml;type=entry
Content-Encoding: compress
```

...compressed payload...

The server rejects request because it only allows the "gzip" content coding:

```
HTTP/1.1 415 Unsupported Media Type
Date: Fri, 09 May 2014 11:43:53 GMT
Accept-Encoding: gzip
Content-Length: 68
Content-Type: text/plain
```

This resource only supports the "gzip" content coding in requests.

...at which point the client can retry the request with the supported "gzip" content coding.

Alternatively, a server that does not support any content codings in requests could answer with:

```
HTTP/1.1 415 Unsupported Media Type
Date: Fri, 09 May 2014 11:43:53 GMT
Accept-Encoding: identity
Content-Length: 61
Content-Type: text/plain
```

This resource does not support content codings in requests.

5. Deployment Considerations

Servers that do not support content codings in requests already are required to fail a request that uses a content coding. Section 6.5.13 of [RFC7231] defines the status code 415 (Unsupported Media Type) for this purpose, so the only change needed is to include the "Accept-Encoding" header field with value "identity" in that response.

Servers that do support some content codings are required to fail requests with unsupported content codings as well. To be compliant with this specification, servers will need to use the status code 415 (Unsupported Media Type) to signal the problem, and will have to include an "Accept-Encoding" header field that enumerates the content codings that are supported. As the set of supported content codings is usually static and small, adding the header field ought to be trivial.

6. Security Considerations

This specification only adds discovery of supported content codings and diagnostics for requests failing due to unsupported content codings. As such, it doesn't introduce any new security considerations over those already present in HTTP/1.1 (Section 9 of [RFC7231]) and HTTP/2 (Section 10 of [RFC7540]).

However, the point of better discoverability and diagnostics is to make it easier to use content codings in requests. This might lead to increased usage of compression codings such as gzip (Section 4.2 of [RFC7230]), which, when used over a secure channel, can enable side-channel attacks such as BREACH (see Section 10.6 of [RFC7540] and [BREACH]). At the time of publication, it was unclear how BREACH-like attacks can be applied to compression in HTTP requests.

7. IANA Considerations

7.1. Header Field Registry

HTTP header fields are registered within the "Message Headers" registry located at

<<http://www.iana.org/assignments/message-headers>>, as defined by [BCP90].

This document updates the definition of the "Accept-Encoding" header field, so the "Permanent Message Header Field Names" registry ought to be updated accordingly:

Header Field Name	Protocol	Status	Reference
Accept-Encoding	http	standard	[RFC7231], Section 5.3.4, and Section 3 of this document

7.2. Status Code Registry

HTTP status codes are registered within the "Status Code" registry located at <<http://www.iana.org/assignments/http-status-codes>>.

This document updates the definition of the status code 415 (Unsupported Media Type), so the "Status Code" registry ought to be updated accordingly:

Value	Description	Reference
415	Unsupported Media Type	[RFC7231], Section 6.5.13, and Section 3 of this document

8. References

8.1. Normative References

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

8.2. Informative References

- [BCP90] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004, <<http://www.rfc-editor.org/info/bcp90>>.
- [BREACH] Gluck, Y., Harris, N., and A. Prado, "BREACH: Reviving the CRIME Attack", July 2013, <<http://breachattack.com/resources/BREACH%20-%20SSL,%20gone%20in%2030%20seconds.pdf>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<http://www.rfc-editor.org/info/rfc7540>>.

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

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

Clarified that the information returned in Accept-Encoding is per resource, not per server.

Added some deployment considerations.

Updated HTTP/1.1 references.

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

Restrict the scope of A-E from "future requests" to "at the time of this request".

Mention use of A-E in responses other than 415.

Recommend not to include A-E in a 415 response unless there was actually a problem related to content coding.

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

First Working Group draft; updated boilerplate accordingly.

A.4. Since draft-ietf-httpbis-cice-00

Apply editorial improvements suggested by Mark Nottingham.

A.5. Since draft-ietf-httpbis-cice-01

Clarify that we're also extending the definition of status code 415 (so update that IANA registry entry as well).

A.6. Since draft-ietf-httpbis-cice-02

Removed normative language that required used of Accept-Encoding in responses (which would have made existing servers non-compliant).

Add BREACH like attacks to security considerations
([<https://github.com/httpwg/http-extensions/issues/94>](https://github.com/httpwg/http-extensions/issues/94)).

Appendix B. Acknowledgements

Thanks go to the members of the and HTTPbis Working Group, namely Amos Jeffries, Ben Campbell, Mark Nottingham, Pete Resnick, Stephen Farrell, and Ted Hardie.

Author's Address

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

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

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

M. Nottingham

M. Thomson
Mozilla
March 17, 2017

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

Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 18, 2017.

Copyright Notice

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

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

Table of Contents

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

1. Introduction

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

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

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

1.1. Goals and Non-Goals

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

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

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

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

1.2. Notational Conventions

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

2. Using HTTP URIs over TLS

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

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

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

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

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

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

2.1. Alternative Server Opt-In

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

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

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

HEADERS

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

HEADERS

```
:status = 200
content-type = application/json
```

DATA

```
+ END_STREAM
[ "http://www.example.com", "http://example.com" ]
```

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

2.2. Interaction with "https" URIs

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

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

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

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

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

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

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

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

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

3. IANA Considerations

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

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

4. Security Considerations

4.1. Security Indicators

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

4.2. Downgrade Attacks

A downgrade attack against the negotiation for TLS is possible.

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

4.3. Privacy Considerations

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

4.4. Confusion Regarding Request Scheme

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

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

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

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

4.5. Server Controls

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

5. References

5.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <<http://www.rfc-editor.org/info/rfc2818>>.

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<http://www.rfc-editor.org/info/rfc5785>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<http://www.rfc-editor.org/info/rfc6454>>.
- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, DOI 10.17487/RFC7232, June 2014, <<http://www.rfc-editor.org/info/rfc7232>>.
- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<http://www.rfc-editor.org/info/rfc7234>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<http://www.rfc-editor.org/info/rfc7540>>.
- [RFC7838] Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <<http://www.rfc-editor.org/info/rfc7838>>.

5.2. Informative References

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

- [RFC7435] Dukhovni, V., "Opportunistic Security: Some Protection Most of the Time", RFC 7435, DOI 10.17487/RFC7435, December 2014, <<http://www.rfc-editor.org/info/rfc7435>>.
- [RFC7469] Evans, C., Palmer, C., and R. Sleevi, "Public Key Pinning Extension for HTTP", RFC 7469, DOI 10.17487/RFC7469, April 2015, <<http://www.rfc-editor.org/info/rfc7469>>.
- [W3C.CR-mixed-content-20160802] West, M., "Mixed Content", World Wide Web Consortium CR CR-mixed-content-20160802, August 2016, <<https://www.w3.org/TR/2016/CR-mixed-content-20160802>>.

Appendix A. Acknowledgements

Mike Bishop contributed significant text to this document.

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

Authors' Addresses

Mark Nottingham

Email: mnot@mnot.net

URI: <https://www.mnot.net/>

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

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

R. Fielding
Adobe Systems Incorporated
M. Nottingham
March 2, 2016

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

Abstract

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

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

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

Note to Readers

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

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any

time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 3, 2016.

Copyright Notice

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

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

Table of Contents

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

1. Introduction

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

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

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

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

1.1. Examples

For example, this response header field:

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

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

This Key:

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

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

A more complex example:

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

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

1.2. Notational Conventions

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

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

2. The "Key" Response Header Field

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

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

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

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

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

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

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

The following header fields have the same effect:

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

However, Key's use of parameters allows:

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

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

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

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

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

2.1. Relationship with Vary

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

For example,

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

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

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

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

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

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

2.2. Calculating a Secondary Cache Key

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

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

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

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

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

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

- 4) Create "key_list" by splitting "key_value" on "," characters, excepting "," characters within quoted strings, as per [RFC7230] Section 3.2.6..
- 5) For "key_item" in "key_list":
 - 1) Remove any leading and trailing WSP from "key_item".
 - 2) If "key_item" does not contain a ";" character, fail parameter processing (Section 2.2.2) and skip to the next "key_item".
 - 3) Let "field_name" be the string before the first ";" character in "key_item", removing any WSP between them.
 - 4) Let "field_value" be the result of Creating a Header Field Value (Section 2.2.1) with "field_name" as the "target_field_name" and the request header list as "header_list".
 - 5) Let "parameters" be the string after the first ";" character in "key_item", removing any WSP between them.
 - 6) Create "param_list" by splitting "parameters" on ";" characters, excepting ";" characters within quoted strings, as per [RFC7230] Section 3.2.6.
 - 7) For "parameter" in "param_list":
 - 1) If "parameter" does not contain a "=", fail parameter processing (Section 2.2.2) and skip to the next "key_item".
 - 2) Remove any WSP at the beginning and/or end of "parameter".
 - 3) Let "param_name" be the string before the first "=" character in "parameter", case-normalized to lowercase.
 - 4) If "param_name" does not identify a Key parameter processing algorithm that is implemented, fail parameter processing (Section 2.2.2) and skip to the next "key_item".
 - 5) Let "param_value" be the string after the first "=" character in "parameter".
 - 6) If the first and last characters of "param_value" are both DQUOTE:
 - 1) Remove the first and last characters of "param_value".
 - 2) Replace quoted-pairs within "param_value" with the octet following the backslash, as per [RFC7230] Section 3.2.6.
 - 7) If "param_value" does not conform to the syntax defined for it by the parameter definition, fail parameter processing Section 2.2.2 and skip to the next "key_item".
 - 8) Run the identified processing algorithm on "field_value" with the "param_value", and append the result to

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

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

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

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

2.2.1. Creating a Header Field Value

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

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

2.2.2. Failing Parameter Processing

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

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

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

2.3. Key Parameters

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

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

2.3.1. div

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

Its value's syntax is:

div = 1*DIGIT

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

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

For example, the Key:

Key: Bar;div=5

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

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

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

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

2.3.2. partition

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

Its value's syntax is:

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

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

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

For example, the Key:

Key: Foo;partition=20:30:40

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

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

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

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

whereas the following would fall into the second segment:

Foo: 20
Foo: 29.999
Foo: 24 , 10

2.3.3. match

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

Its value's syntax is:

match = (token / quoted-string)

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

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

For example, the Key:

Key: Baz;match="charlie"

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

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

and "0" for these:

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

2.3.4. substr

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

Its value's syntax is:

substr = (token / quoted-string)

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

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

For example, the Key:

Key: Abc;substr=bennet

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

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

and "0" for these:

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

2.3.5. param

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

Its value's syntax is:

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

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

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

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

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

For example, the Key:

Key: Def;param=liam

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

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

3. IANA Considerations

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

3.1. Procedure

Key Parameter registrations MUST include the following fields:

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

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

3.2. Registrations

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

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

4. Security Considerations

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

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

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

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

5. References

5.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.

- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<http://www.rfc-editor.org/info/rfc7231>>.
- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <<http://www.rfc-editor.org/info/rfc7234>>.

5.2. Informative References

- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<http://www.rfc-editor.org/info/rfc6265>>.

Appendix A. Acknowledgements

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

Appendix B. Changes

B.1. Since -00

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

Authors' Addresses

Roy T. Fielding
Adobe Systems Incorporated

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

Mark Nottingham

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

HTTP Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 13, 2016

T. Bray
Textuality
November 10, 2015

An HTTP Status Code to Report Legal Obstacles
draft-ietf-httpbis-legally-restricted-status-04

Abstract

This document specifies a Hypertext Transfer Protocol (HTTP) status code for use when resource access is denied as a consequence of legal demands.

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 [1].

Working Group information can be found at [2] and [3]; source code and issues list for this draft can be found at [4].

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 13, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Requirements	2
3. 451 Unavailable For Legal Reasons	3
4. Identifying Blocking Entities	3
5. Security Considerations	4
6. IANA Considerations	4
7. References	5
Appendix A. Acknowledgements	5
Author's Address	5

1. Introduction

This document specifies a Hypertext Transfer Protocol (HTTP) status code for use when a server operator has received a legal demand to deny access to a resource or to a set of resources which includes the requested resource.

This status code can be used to provide transparency in circumstances where issues of law or public policy affect server operations. This transparency may be beneficial both to these operators and to end users.

[RFC4924] discusses the forces working against transparent operation of the Internet; these clearly include legal interventions to restrict access to content. As that document notes, and as Section 4 of [RFC4084] states, such restrictions should be made explicit.

Feedback should occur on the ietf-http-wg@w3.org mailing list.

2. Requirements

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].

3. 451 Unavailable For Legal Reasons

This status code indicates that the server is denying access to the resource as a consequence of a legal demand.

The server in question might not be an origin server. This type of legal demand typically most directly affects the operations of ISPs and search engines.

Responses using this status code SHOULD include an explanation, in the response body, of the details of the legal demand: the party making it, the applicable legislation or regulation, and what classes of person and resource it applies to. For example:

HTTP/1.1 451 Unavailable For Legal Reasons

Link: <https://spqr.example.org/legislation>; rel="blocked-by"

Content-Type: text/html

```
<html>
  <head><title>Unavailable For Legal Reasons</title></head>
  <body>
    <h1>Unavailable For Legal Reasons</h1>
    <p>This request may not be serviced in the Roman Province
      of Judea due to the Lex Julia Majestatis, which disallows
      access to resources hosted on servers deemed to be
      operated by the People's Front of Judea.</p>
  </body>
</html>
```

The use of the 451 status code implies neither the existence nor non-existence of the resource named in the request. That is to say, it is possible that if the legal demands were removed, a request for the resource still might not succeed.

Note that in many cases clients can still access the denied resource by using technical countermeasures such as a VPN or the Tor network.

A 451 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls; see [RFC7234].

4. Identifying Blocking Entities

As noted above, when an attempt to access a resource fails with status 451, the entity blocking access might or might not be the origin server. There are a variety of entities in the resource-

access path which could choose to deny access, for example ISPs, cache providers, and DNS servers.

It is useful, when legal blockages occur, to be able to identify the entities actually implementing the blocking.

When an entity blocks access to a resource and returns status 451, it SHOULD include a "Link" HTTP header field [RFC5988] whose value is a URI reference [RFC3986] identifying itself. When used for this purpose, the "Link" header field MUST have a "rel" parameter whose value is "blocked-by".

The intent is that the header be used to identify the entity actually implementing blockage, not any other entity mandating it. A human readable response body, as discussed above, is the appropriate location for discussion of administrative and policy issues.

5. Security Considerations

Clients cannot rely upon the use of the 451 status code. It is possible that certain legal authorities might wish to avoid transparency, and not only demand the restriction of access to certain resources, but also avoid disclosing that the demand was made.

6. IANA Considerations

The HTTP Status Codes Registry should be updated with the following entry:

- o Code: 451
- o Description: Unavailable for Legal Reasons
- o Specification: [this document]

The Link Relation Type Registry should be updated with the following entry:

- o Relation Name: blocked-by
- o Description: Identifies the entity blocking access to a resource following on receipt of a legal demand.
- o Reference: This document

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<http://www.rfc-editor.org/info/rfc5988>>.
- [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>>.

7.2. Informative References

- [RFC4084] Klensin, J., "Terminology for Describing Internet Connectivity", BCP 104, RFC 4084, DOI 10.17487/RFC4084, May 2005, <<http://www.rfc-editor.org/info/rfc4084>>.
- [RFC4924] Aboba, B., Ed. and E. Davies, "Reflections on Internet Transparency", RFC 4924, DOI 10.17487/RFC4924, July 2007, <<http://www.rfc-editor.org/info/rfc4924>>.

Appendix A. Acknowledgements

Thanks to Terence Eden, who observed that the existing status code 403 was not really suitable for this situation, and suggested the creation of a new status code.

Thanks also to Ray Bradbury.

Author's Address

Tim Bray
Textuality

Email: tbray@textuality.com
URI: <http://www.tbray.org/ongoing/>

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 3, 2016

M. Nottingham
E. Nygren
Akamai
January 31, 2016

The ORIGIN HTTP/2 Frame
draft-nottingham-httpbis-origin-frame-01

Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 3, 2016.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
1.1. Notational Conventions	2
1.2. The ORIGIN HTTP/2 Frame	2
2. Security Considerations	3
3. Normative References	3
Authors' Addresses	4

1. Introduction

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

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

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].

1.2. The ORIGIN HTTP/2 Frame

The ORIGIN HTTP/2 frame ([RFC7540], Section 4) indicates what origin(s) [RFC6454] the sender considers this connection authoritative for (in the sense of [RFC7540], Section 10.1).

The ORIGIN frame is a non-critical extension to HTTP/2. Endpoints that do not support this frame can safely ignore it.

It MUST occur on stream 0; an ORIGIN frame on any other stream is invalid and MUST be ignored.

When received by a client, it can be used to inform HTTP/2 connection coalescing (see [RFC7540], Section 9.1.1), but does not relax the requirement there that the server is authoritative.

If multiple ORIGIN frames are received on the same connection, only the most recent is to be considered current.

Once an ORIGIN frame has been received and processed, clients that implement this specification SHOULD NOT use that connection for a given origin if it did not appear within the current ORIGIN frame.

The ORIGIN frame type is 0xb (decimal 11).

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

The ORIGIN frame contains the following fields, sets of which may be repeated within the frame to indicate multiple origins:

Origin-Len: An unsigned, 16-bit integer indicating the length, in octets, of the Origin field. Origin: An optional sequence of characters containing the ASCII serialization of an origin ([RFC6454], Section 6.2) that the sender believes this connection is authoritative for.

The ORIGIN frame does not define any flags. It can contain one or more Origin-Len/Origin pairs.

The ORIGIN frame is processed hop-by-hop. An intermediary must not forward ORIGIN frames.

Clients configured to use a proxy MUST ignore any ORIGIN frames received from it.

2. Security Considerations

Clients that blindly trust the ORIGIN frame's contents will be vulnerable to a large number of attacks; hence the reinforcement that this specification does not relax the requirement for server authority in [RFC7540], Section 10.1.

3. 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>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<http://www.rfc-editor.org/info/rfc6454>>.

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

Authors' Addresses

Mark Nottingham
Akamai

Email: mnot@mnot.net
URI: <http://www.mnot.net/>

Erik Nygren
Akamai

Email: nygren@akamai.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 24 October 2021

J. F. Reschke
greenbytes
22 April 2021

A JSON Encoding for HTTP Field Values
draft-reschke-http-jfv-14

Abstract

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

Editorial Note

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

Distribution of this document is unlimited. Although this is not a work item of the HTTPbis Working Group, comments should be sent to the Hypertext Transfer Protocol (HTTP) mailing list at ietf-http-wg@w3.org (<mailto:ietf-http-wg@w3.org>), which may be joined by sending a message with subject "subscribe" to ietf-http-wg-request@w3.org (<mailto:ietf-http-wg-request@w3.org?subject=subscribe>).

Discussions of the HTTPbis Working Group are archived at <http://lists.w3.org/Archives/Public/ietf-http-wg/>.

XML versions and latest edits for this document are available from <http://greenbytes.de/tech/webdav/#draft-reschke-http-jfv>.

The changes in this draft are summarized in Appendix D.17.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <https://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 24 October 2021.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	3
1.1. Relation to "Structured Field Values for HTTP" (RFC8941)	4
2. Data Model and Format	4
3. Sender Requirements	5
3.1. Example	5
4. Recipient Requirements	6
4.1. Example	6
5. Using this Format in Field Definitions	7
6. Deployment Considerations	7
7. Interoperability Considerations	7
7.1. Encoding and Characters	7
7.2. Numbers	8
7.3. Object Constraints	8
8. Internationalization Considerations	8
9. Security Considerations	8
10. References	9
10.1. Normative References	9
10.2. Informative References	9
10.3. Specifications Using This Syntax (at some point of time)	10
Appendix A. Comparison with Structured Fields	10
A.1. Base Types	10
A.2. Structures	11
Appendix B. Use of JSON Field Value Encoding in the Wild	11
B.1. W3C Reporting API Specification	12
B.2. W3C Clear Site Data Specification	12
B.3. W3C Feature Policy Specification	12
Appendix C. Implementations	12
Appendix D. Change Log	12

D.1.	Since draft-reschke-http-jfv-00	12
D.2.	Since draft-reschke-http-jfv-01	12
D.3.	Since draft-reschke-http-jfv-02	13
D.4.	Since draft-reschke-http-jfv-03	13
D.5.	Since draft-reschke-http-jfv-04	13
D.6.	Since draft-ietf-httpbis-jfv-00	13
D.7.	Since draft-ietf-httpbis-jfv-01	13
D.8.	Since draft-ietf-httpbis-jfv-02	13
D.9.	Since draft-reschke-http-jfv-05	13
D.10.	Since draft-reschke-http-jfv-06	14
D.11.	Since draft-reschke-http-jfv-07	14
D.12.	Since draft-reschke-http-jfv-08	14
D.13.	Since draft-reschke-http-jfv-09	14
D.14.	Since draft-reschke-http-jfv-10	14
D.15.	Since draft-reschke-http-jfv-11	14
D.16.	Since draft-reschke-http-jfv-12	15
D.17.	Since draft-reschke-http-jfv-13	15
Acknowledgements	15
Author's Address	15

1. Introduction

Defining syntax for new HTTP fields ([HTTP], Section 5) is non-trivial. Among the commonly encountered problems are:

- * There is no common syntax for complex field values. Several well-known fields do use a similarly looking syntax, but it is hard to write generic parsing code that will both correctly handle valid field values but also reject invalid ones.
- * The HTTP message format allows field lines to repeat, so field syntax needs to be designed in a way that these cases are either meaningful, or can be unambiguously detected and rejected.
- * HTTP does not define a character encoding scheme ([RFC6365], Section 2), so fields are either stuck with US-ASCII ([RFC0020]), or need out-of-band information to decide what encoding scheme is used. Furthermore, APIs usually assume a default encoding scheme in order to map from octet sequences to strings (for instance, [XMLHttpRequest] uses the IDL type "ByteString", effectively resulting in the ISO-8859-1 character encoding scheme [ISO-8859-1] being used).

(See Section 16.3 of [HTTP] for a summary of considerations for new fields.)

This specification addresses the issues listed above by defining both a generic JSON-based ([RFC8259]) data model and a concrete wire format that can be used in definitions of new fields, where the goals were:

- * to be compatible with field recombination when field lines occur multiple times in a single message (Section 5.3 of [HTTP]), and
- * not to use any problematic characters in the field value (non-ASCII characters and certain whitespace characters).

1.1. Relation to "Structured Field Values for HTTP" ([RFC8941])

"Structured Field Values for HTTP", an IETF RFC on the Standards Track, is a different approach to this set of problems. It uses a more compact notation, similar to what is used in existing header fields, and avoids several potential interoperability problems inherent to the use of JSON.

In general, that format is preferred for newly defined fields. The JSON-based format defined by this document might however be useful in case the data that needs to be transferred is already in JSON format, or features not covered by "Structured Field Values" are needed.

See Appendix A for more details.

2. Data Model and Format

In HTTP, field lines with the same field name can occur multiple times within a single message (Section 5.3 of [HTTP]). When this happens, recipients are allowed to combine the field line values using commas as delimiter, forming a combined "field value". This rule matches nicely JSON's array format (Section 5 of [RFC8259]). Thus, the basic data model used here is the JSON array.

Field definitions that need only a single value can restrict themselves to arrays of length 1, and are encouraged to define error handling in case more values are received (such as "first wins", "last wins", or "abort with fatal error message").

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

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

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

Characters in JSON strings that are not allowed or discouraged in HTTP field values – that is, not in the "VCHAR" definition – need to be represented using JSON's "backslash" escaping mechanism ([RFC8259], Section 7).

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

Formally, using the HTTP specification's ABNF extensions defined in Section 5.6.1 of [HTTP]:

```
json-field-value = #json-field-item
json-field-item  = JSON-Text
                  ; see [RFC8259], Section 2,
                  ; post-processed so that only VCHAR characters
                  ; are used
```

3. Sender Requirements

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

1. generating the JSON representation,
2. stripping all JSON control characters (CR, HTAB, LF), or replacing them by space ("SP") characters,
3. replacing all remaining non-VSPACE characters by the equivalent backslash-escape sequence ([RFC8259], Section 7).

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

3.1. Example

With the JSON data below, containing the non-ASCII characters "ü" (LATIN SMALL LETTER U WITH DIAERESIS, U+00FC) and "€" (EURO SIGN, U+20AC):

```
[
  {
    "destination": "Münster",
    "price": 123,
    "currency": ""
  }
]
```

The generated field value would be:

```
{ "destination": "M\u00FCnster", "price": 123, "currency": "\u20AC" }
```

4. Recipient Requirements

To map a set of HTTP field line values to a JSON array:

1. combine all field line values into a single field value as per Section 5.3 of [HTTP],
2. add a leading begin-array ("[") octet and a trailing end-array ("]") octet, then
3. run the resulting octet sequence through a JSON parser.

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

4.1. Example

An HTTP message containing the field lines:

```
Example: "\u221E"
Example: {"date":"2012-08-25"}
Example: [17,42]
```

would be parsed into the JSON array below:

```
[
  "",
  {
    "date": "2012-08-25"
  },
  [
    17,
    42
  ]
]
```

5. Using this Format in Field Definitions

Specifications defining new HTTP fields need to take the considerations listed in Section 16.3 of [HTTP] into account. Many of these will already be accounted for by using the format defined in this specification.

Readers of HTTP-related specifications frequently expect an ABNF definition of the field value syntax. This is not really needed here, as the actual syntax is JSON text, as defined in Section 2 of [RFC8259].

A very simple way to use this JSON encoding thus is just to cite this specification – specifically the "json-field-value" ABNF production defined in Section 2 – and otherwise not to talk about the details of the field syntax at all.

An alternative approach is just to repeat the ABNF-related parts from Section 2.

This frees the specification from defining the concrete on-the-wire syntax. What's left is defining the field value in terms of a JSON array. An important aspect is the question of extensibility, e.g. how recipients ought to treat unknown field names. In general, a "must ignore" approach will allow protocols to evolve without versioning or even using entire new field names.

6. Deployment Considerations

This JSON-based syntax will only apply to newly introduced fields, thus backwards compatibility is not a problem. That being said, it is conceivable that there is existing code that might trip over double quotes not being used for HTTP's quoted-string syntax (Section 5.6.4 of [HTTP]).

7. Interoperability Considerations

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

7.1. Encoding and Characters

This specification requires that field values use only US-ASCII characters, and thus by definition uses a subset of UTF-8 (Section 2.1 of [RFC7493]).

Furthermore, escape sequences in JSON strings (Section 7 of [RFC8259]) - both in object member names and string values - are not allowed to represent non-Unicode code points such as unpaired surrogates or Noncharacters (see "General Structure" in [UNICODE]).

7.2. Numbers

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

7.3. Object Constraints

As described in Section 4 of [RFC8259], JSON parser implementations differ in the handling of duplicate object names. Therefore, senders are not allowed to use duplicate object names, and recipients are advised to either treat field values with duplicate names as invalid (consistent with [RFC7493], Section 2.3) or use the lexically last value (consistent with [ECMA-262], Section 24.3.1.1).

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

8. Internationalization Considerations

In current versions of HTTP, field values are represented by octet sequences, usually used to transmit ASCII characters, with restrictions on the use of certain control characters, and no associated default character encoding, nor a way to describe it ([HTTP], Section 5).

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

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

9. Security Considerations

Using JSON-shaped field values is believed to not introduce any new threads beyond those described in Section 12 of [RFC8259], namely the risk of recipients using the wrong tools to parse them.

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

10. References

10.1. Normative References

- [HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", Work in Progress, Internet-Draft, draft-ietf-httpbis-semantics-15, 30 March 2021, <<https://tools.ietf.org/html/draft-ietf-httpbis-semantics-15>>.
- [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>>.
- [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>>.
- [RFC7493] Bray, T., Ed., "The I-JSON Message Format", RFC 7493, DOI 10.17487/RFC7493, March 2015, <<https://www.rfc-editor.org/info/rfc7493>>.
- [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 8259, DOI 10.17487/RFC8259, December 2017, <<https://www.rfc-editor.org/info/rfc8259>>.
- [UNICODE] The Unicode Consortium, "The Unicode Standard", <<http://www.unicode.org/versions/latest/>>.

10.2. Informative References

- [ECMA-262] Ecma International, "ECMA-262 6th Edition, The ECMAScript 2015 Language Specification", Standard ECMA-262, June 2015, <<http://www.ecma-international.org/ecma-262/6.0/>>.
- [ISO-8859-1] International Organization for Standardization, "Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1", ISO/IEC 8859-1:1998, 1998.

- [RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, DOI 10.17487/RFC6365, September 2011, <<https://www.rfc-editor.org/info/rfc6365>>.
- [RFC8941] Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <<https://www.rfc-editor.org/info/rfc8941>>.
- [XMLHttpRequest] WhatWG, "XMLHttpRequest", <<https://xhr.spec.whatwg.org/>>.

10.3. Specifications Using This Syntax (at some point of time)

- [CLEARSITE] West, M., "Clear Site Data", W3C Working Draft WD-clear-site-data-20171130, 30 November 2017, <<https://www.w3.org/TR/2017/WD-clear-site-data-20171130/>>. Latest version available at <<https://www.w3.org/TR/clear-site-data/>>.
- [FEATUREPOL] Clelland, I., "Feature Policy", W3C Editor's Draft , <<https://w3c.github.io/webappsec-feature-policy/>>.
- [REPORTING] Creager, D., Grigorik, I., Meyer, P., and M. West, "Reporting API", W3C Working Draft WD-reporting-1-20180925, 25 September 2018, <<https://www.w3.org/TR/2018/WD-reporting-1-20180925/>>. Latest version available at <<https://www.w3.org/TR/reporting-1/>>.

Appendix A. Comparison with Structured Fields

A.1. Base Types

Type	in Structured Fields	in JSON-based Fields
Integer	[RFC8941], Section 3.3.1 (restricted to 15 digits)	[RFC8259], Section 6
Decimal	[RFC8941], Section 3.3.2	[RFC8259], Section 6

	(a fixed point decimal restricted to 12 + 3 digits)	
String	[RFC8941], Section 3.3.3	[RFC8259], Section 7
	(only ASCII supported, non-ASCII requires using Byte Sequences)	
Token	[RFC8941], Section 3.3.4	not available
Byte Sequence	[RFC8941], Section 3.3.5	not available
		(usually mapped to strings using base64 encoding)
Boolean	[RFC8941], Section 3.3.6	[RFC8259], Section 3

Table 1

Structured Fields provide more data types (such as "token" or "byte sequence"). Numbers are restricted, avoiding the JSON interop problems described in Section 7.2. Strings are limited to ASCII, requiring the use of byte sequences should non-ASCII characters be needed.

A.2. Structures

Structured Fields define Lists ([RFC8941], Section 3.1), similar to JSON arrays ([RFC8259], Section 5), and Dictionaries ([RFC8941], Section 3.2), similar to JSON objects ([RFC8259], Section 4).

In addition, most items in Structured Fields can be parametrized ([RFC8941], Section 3.1.2), attaching a dictionary-like structure to the value. To emulate this in JSON based field, an additional nesting of objects would be needed.

Finally, nesting of data structures is intentionally limited to two levels (see Appendix A.1 of [RFC8941] for the motivation).

Appendix B. Use of JSON Field Value Encoding in the Wild

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

Since work started on this document, various specifications have adopted this format. At least one of these moved away after the HTTP Working Group decided to focus on [RFC8941] (see thread starting at <<https://lists.w3.org/Archives/Public/ietf-http-wg/2016OctDec/0505.html>>).

The sections below summarize the current usage of this format.

B.1. W3C Reporting API Specification

Defined in W3C Working Draft "Reporting API" (Section 3.1 of [REPORTING]). Still in use in latest working draft dated September 2018.

B.2. W3C Clear Site Data Specification

Used in earlier versions of "Clear Site Data". The current version replaces the use of JSON with a custom syntax that happens to be somewhat compatible with an array of JSON strings (see Section 3.1 of [CLEARSITE] and <<https://lists.w3.org/Archives/Public/ietf-http-wg/2017AprJun/0214.html>> for feedback).

B.3. W3C Feature Policy Specification

Originally defined in W3C document "Feature Policy" ([FEATUREPOL]), but switched to use of Structured Header Fields ([RFC8941]).

Appendix C. Implementations

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

See <<https://github.com/reschke/json-fields>> for a proof-of-concept (in development).

Appendix D. Change Log

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

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

Editorial fixes + working on the TODOs.

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

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

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

Mention Kazuho Oku's proposal for abbreviated forms.

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

Expand I18N section.

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

Mention relation to KEY header field.

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

Between June and December 2016, this was a work item of the HTTP working group (see <<https://datatracker.ietf.org/doc/draft-ietf-httpbis-jfv/>>). Work (if any) continues now on <<https://datatracker.ietf.org/doc/draft-reschke-http-jfv/>>.

Changes made while this was a work item of the HTTP Working Group:

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

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

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

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

D.8. Since draft-ietf-httpbis-jfv-02

Move non-essential parts into appendix.

Updated XHR reference.

D.9. Since draft-reschke-http-jfv-05

Add meat to "Using this Format in Header Field Definitions".

Add a few lines on the relation to "Key".

Summarize current use of the format.

D.10. Since draft-reschke-http-jfv-06

RFC 5987 is obsoleted by RFC 8187.

Update CLEARSITE comment.

D.11. Since draft-reschke-http-jfv-07

Update JSON and HSTRUCT references.

FEATUREPOL doesn't use JSON syntax anymore.

D.12. Since draft-reschke-http-jfv-08

Update HSTRUCT reference.

Update notes about CLEARSITE and FEATUREPOL.

D.13. Since draft-reschke-http-jfv-09

Update HSTRUCT and FEATUREPOL references.

Update note about REPORTING.

Changed category to "informational".

D.14. Since draft-reschke-http-jfv-10

Update HSTRUCT reference.

D.15. Since draft-reschke-http-jfv-11

Update HSTRUCT reference.

Update note about FEATUREPOL (now using Structured Fields).

Reference [HTTP] instead of RFC723* and adjust (header) field terminology accordingly.

Remove discussion about the relation to KEY (as that spec is dormant: <https://datatracker.ietf.org/doc/draft-ietf-httpbis-key/>).

Remove appendices "Examples" and "Discussion".

Mark "Use of JSON Field Value Encoding in the Wild" for removal in RFC.

D.16. Since draft-reschke-http-jfv-12

Update HTTP reference and update terminology some more.

Update HSTRUCT reference (now RFC 8941).

D.17. Since draft-reschke-http-jfv-13

Update HTTP reference.

Mention test implementation.

Clarify that Unicode unpaired surrogates or Noncharacters must not be sent.

Rewrite text about [RFC8941], add appendix comparing both formats.

And send/receive examples.

Acknowledgements

Thanks go to the Hypertext Transfer Protocol Working Group participants.

Author's Address

Julian F. Reschke
greenbytes GmbH
Hafenweg 16
48155 Münster
Germany

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

Httpbis
Internet-Draft
Intended status: Experimental
Expires: April 18, 2016

H. Ruellan
Y. Fablet
R. Bellessort
F. Denoual
F. Maze
Canon CRF
October 16, 2015

Accept-Push-Policy Header Field
draft-ruellan-http-accept-push-policy-00

Abstract

The "Accept-Push-Policy" and "Push-Policy" header fields enable a client and a server to negotiate the behaviour of the server regarding the usage of push on a per-request basis.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 18, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. Terminology	3
2. Push Control Use Cases	3
2.1. Adapting Push Behaviour	3
2.2. Load Balancer	4
2.3. DASH Fast Start	4
2.4. Fast Page Load	5
2.5. Use Cases Requirements	5
3. Push Policy	5
3.1. The Accept-Push-Policy Header Field	6
3.2. Push-Policy Header Field	6
3.3. Push Policy Values	7
3.3.1. None Push Policy	7
3.3.2. Head Push Policy	7
3.3.3. Default Push Policy	8
3.3.4. Fast-Load Push Policy	8
4. IANA Considerations	8
5. Security Considerations	9
6. References	9
6.1. Normative References	9
6.2. Informative References	9
Authors' Addresses	10

1. Introduction

HTTP/2 [RFC7540], the new version of the HTTP protocol, not only provides significant improvements compared to HTTP/1.1 (see [RFC7230] and [RFC7231]), but also provides several new features. Among these is Server Push, which enables a server to send responses to a client without having received the corresponding requests.

The range of possibilities offered by Server Push is a new domain wide open for experimentation. A first usage was foreseen early in the addition of this feature into HTTP/2, which is to replace the inlining of sub-resources inside a main resource, by pushing these sub-resources in response to the request for the main resource. As

described in [HighPerformance], with HTTP/1.1 a web designer may want to optimize the page load time by packing a whole web page into a single HTTP response. This can be achieved by inlining the CSS, JavaScript, and images inside the HTML document. By removing the need for the client to send requests for these sub-resources, this inlining technique can reduce the page load time by roughly a RTT. With HTTP/2, the same results can be obtained by pushing the sub-resources instead of inlining them. Using push has the advantage of keeping each sub-resource independent.

HTTP/2 provides a few ways of controlling Server Push from the client side. First, the SETTINGS parameter "SETTINGS_ENABLE_PUSH" allows a client to globally enable or disable push on a HTTP/2 connection. In addition, HTTP/2 Flow Control can be used to limit the bandwidth used by pushed resources.

These options provide only a coarse control of the usage of Server Push from the client side. In some cases, a more fine-grained control would be useful. This document describes several use cases where controlling Server Push would be useful for the client. It then proposes new header fields for realizing this control.

1.1. Terminology

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

This document uses the Augmented BNF defined in [RFC5234].

2. Push Control Use Cases

2.1. Adapting Push Behaviour

A browser may want to ask the server to adapt its behaviour for pushing resources depending on the user's actions. For example, after navigating through a site for some time, the browser may have many sub-resources in its cache and may prefer that the server doesn't push sub-resources anymore to prevent wasting bandwidth. This could be further optimized with the browser asking the server to push only response metadata (i.e., the responses pushed by the server correspond to requests made with the HEAD method instead of requests made with the GET method). By receiving in advance the list of sub-resources corresponding to a specific request, the browser would be able to fetch early on any missing sub-resource.

As another example, when a user opens many pages on the same site, the browser may want to receive pushed sub-resources only for the foreground tab and not for any background tab. This results in a better optimization of the page load time for the tab that is visible to the user.

2.2. Load Balancer

A second use case is a load balancer serving both HTTP/1.1 and HTTP/2 clients, and using HTTP/2 to connect to the backend servers, as described in [LoadBalancer].

The load balancer uses the same HTTP/2 connection towards a backend server to forward the requests received from several clients. When the client is a HTTP/1.1 client, the load balancer doesn't want the backend server to push any resource in response to the client's request. On the contrary, when the client is a HTTP/2 client, the load balancer would like the backend server to push sub-resources associated to the client's request.

The load balancer would like to be able to enable or disable push on a per-request basis. This would enable it to optimize the server behaviour depending on the client's capacity.

2.3. DASH Fast Start

Controlling the server behaviour regarding push may also be useful for specific applications. As an example, DASH [DASH] is a technology for streaming media content over HTTP. The media content is split into small file-based segments that can be retrieved through HTTP requests. Potentially, the media content is made available with different quality levels. A media presentation description (MPD) describes the organization of the media.

To render a media, a DASH client needs to first download the MPD, process it, and then request the necessary media segments. When requesting a MPD to play the associated media content, it would be useful for a DASH client to be able to ask the server to push some initial content (for example, the initialization segments, and possibly the first content segments).

However, there are also cases when it is not useful for the DASH client to receive in advance this initial content. For example, in a video program guide, the DASH client may want to download several MPDs corresponding to different media content, but doesn't want to receive the initial content for all of these. Therefore, it is useful for the DASH client to be able to specify in a request for a MPD whether it wants the server to push some initial content.

In addition, when the DASH client asks the server to push some initial content, it could be useful for it to have some feedback from the server. This feedback would indicate whether the server is intending to push this initial content. The client could adapt its behaviour depending on this indication. For example, the client could start rendering the media sooner if it knows that the server is pushing the initial content.

2.4. Fast Page Load

The previous use case can be expanded to the more generic use case of downloading quickly a web page. As described in [Breaking1000msBarrier], it is important for the user perception to keep the perceived latency of loading a web page under 1000 ms. This can be difficult when using a mobile connection with a high latency. Part of the solution proposed in [Breaking1000msBarrier] for HTTP/1.1 is to inline all the sub-resources necessary for achieving a first rendering of the web page. With HTTP/2, the inlining of these sub-resources can be replaced by having the server push them.

Therefore, a client detecting that it is using a high-latency network could improve the user perceived latency by asking the server to push all the sub-resources necessary for a first display of a web page.

2.5. Use Cases Requirements

The analysis of these use cases enables to build a list of requirements for defining a fine-grained control over the usage of push by a server.

- o The client can ask the server not to push any resource in response to a request.
- o The client can ask the server to only push response metadata.
- o The client can ask the server to use an application-defined behaviour regarding push.
- o The server can indicate to the client its behaviour regarding push when processing a request.

3. Push Policy

A `_push policy_` defines the server behaviour regarding push when processing a request. Different push policies can be used when processing different requests.

This section defines new header fields enabling a client and a server to negotiate the push policy used by the server to process a given request.

The new "Accept-Push-Policy" header field enables a client to express its expectations regarding the server's push policy for processing a request.

The "Push-Policy" header field enables a server to state the push policy used when processing a request.

3.1. The Accept-Push-Policy Header Field

A client can express the desired push policy for a request by sending an "Accept-Push-Policy" header field in the request.

Accept-Push-Policy = token ; a push policy name

The header field value contains the push policy that the client expects the server to use when processing the request.

Possibly, the "Accept-Push-Policy" header field could be extended to support carrying multiple policies, as a comma-separated list of tokens. The server could choose its preferred policy among those proposed by the client.

3.2. Push-Policy Header Field

A server can indicate to a client the push policy it used when processing a request by sending a "Push-Policy" header field in the corresponding response.

Push-Policy = token ; a push policy name

The server MUST follow the indicated push policy when processing the client request associated to the response.

The "Push-Policy" header field can be used as an acknowledgement from the server after receiving a request containing the "Accept-Push-Policy" header field.

If the "Accept-Push-Policy" header field can contain a list of push policy names, the "Push-Policy" header field can be used to express which push policy was selected by the server.

The server can also choose a push policy not corresponding to the client's expectation as expressed in the "Accept-Push-Policy" header, and specify the selected push policy in the "Push-Policy" header field.

3.3. Push Policy Values

This section defines some generic push policies. Other push policies can be standardized for either a generic usage, or for an application-specific usage. In addition, private push policies can be used by a web application.

TBD: select the form of private push policies (URN, "X-" values...).

3.3.1. None Push Policy

The "None" push policy value indicates that no resource is pushed when processing a request.

For example, a browser sending a request for a background tab could ask the server not to push any resources in response to this request by sending an "Accept-Push-Policy" header with the "None" value. This would result in the following HTTP/2 header block:

```
:method = GET
:scheme = https
:path = /index.html
host = example.org
accept = text/html
accept-push-policy = none
```

3.3.2. Head Push Policy

The "Head" push policy value indicates that only response metadata are pushed (the server is pushing responses corresponding to requests made with the HEAD method).

For example, a browser may already have many resources from a web site in its cache. It could ask the server to push only response metadata. This would allow the browser to know early on the resources useful for rendering a web page (i.e., before receiving and parsing the HTML document), without taking the risk of wasting bandwidth with resources already in its cache. In this example, the browser's request would contain the following HTTP/2 header block:

```
:method = GET
:scheme = https
```

```
:path = /index.html
host = example.org
accept = text/html
accept-push-policy = head
```

3.3.3. Default Push Policy

The "Default" push policy value indicates that the server is using its default behaviour for pushing resources when processing a request.

For example, a server not fulfilling a client's expectation regarding the push policy could indicate this with the "Default" push policy. It would send the following HTTP/2 header block in its response:

```
:status 200
push-policy = default
```

3.3.4. Fast-Load Push Policy

The "Fast-Load" push policy value indicates that the sub-resources necessary for a first rendering of a main resource are pushed alongside the response containing this main resource.

A server using the "Fast-Load" push policy while processing a request can push sub-resources not necessary for a first rendering, but SHOULD prioritize sub-resources necessary for this first rendering.

For example, a client detecting that it is using a high-latency network can try to improve the user perceived latency by asking the server to push the sub-resources necessary for a first rendering of a main page by including an "Accept-Push-Policy" header with the "Fast-Load" value. This would result in the following HTTP/2 header block:

```
:method = GET
:scheme = https
:path = /index.html
host = example.org
accept = text/html
accept-push-policy = fast-load
```

4. IANA Considerations

TBD

5. Security Considerations

TBD

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<http://www.rfc-editor.org/info/rfc7540>>.

6.2. Informative References

- [Breaking1000msBarrier] Grigorik, I., "Breaking the 1000 ms mobile barrier", November 2013.
- [DASH] "Dynamic adaptive streaming over HTTP (DASH)", ISO/IEC: 23009-1:2014 , 2014.
- [HighPerformance] Grigorik, I., "High Performance Browser Networking", September 2013.
- [LoadBalancer] Douglas, S., "PUSH_PROMISE and load balancers", September 2014, <<https://lists.w3.org/Archives/Public/ietf-http-wg/2014JulSep/2374.html>>.
- [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>>.

Authors' Addresses

Herve Ruellan
Canon CRF

Email: herve.ruellan@crf.canon.fr

Youenn Fablet
Canon CRF

Email: youenn.fablet@crf.canon.fr

Romain Bellessort
Canon CRF

Email: romain.bellessort@crf.canon.fr

Franck Denoual
Canon CRF

Email: franck.denoual@crf.canon.fr

Frederic Maze
Canon CRF

Email: frederic.maze@crf.canon.fr

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

M. Thomson
Mozilla
October 19, 2015

Encrypted Content-Encoding for HTTP
draft-thomson-http-encryption-02

Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 21, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
1.1. Notational Conventions	3
2. The "aesgcm128" HTTP Content Encoding	3
3. The Encryption HTTP Header Field	5
3.1. Encryption Header Field Parameters	6
3.2. Content Encryption Key Derivation	6
3.3. Nonce Derivation	7
4. Encryption-Key Header Field	7
4.1. Explicit Key	8
4.2. Diffie-Hellman	8
5. Examples	9
5.1. Successful GET Response	9
5.2. Encryption and Compression	9
5.3. Encryption with More Than One Key	9
5.4. Encryption with Explicit Key	10
5.5. Diffie-Hellman Encryption	10
6. Security Considerations	11
6.1. Key and Nonce Reuse	11
6.2. Content Integrity	12
6.3. Leaking Information in Headers	12
6.4. Poisoning Storage	13
6.5. Sizing and Timing Attacks	13
7. IANA Considerations	13
7.1. The "aesgcm128" HTTP Content Encoding	13
7.2. Encryption Header Fields	13
7.3. The HTTP Encryption Parameter Registry	14
7.3.1. keyid	14
7.3.2. salt	14
7.3.3. rs	15
7.4. The HTTP Encryption-Key Parameter Registry	15
7.4.1. keyid	15
7.4.2. aesgcm128	15
7.4.3. dh	16
8. References	16
8.1. Normative References	16
8.2. Informative References	17
Appendix A. JWE Mapping	18
Appendix B. Acknowledgements	19
Author's Address	19

1. Introduction

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

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

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

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

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

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

This mechanism is likely only a small part of a larger design that uses content encryption. How clients and servers acquire and identify keys will depend on the use case. Though a complete key management system is not described, this document defines an Encryption-Key header field that can be used to convey keying material.

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 "aesgcm128" HTTP Content Encoding

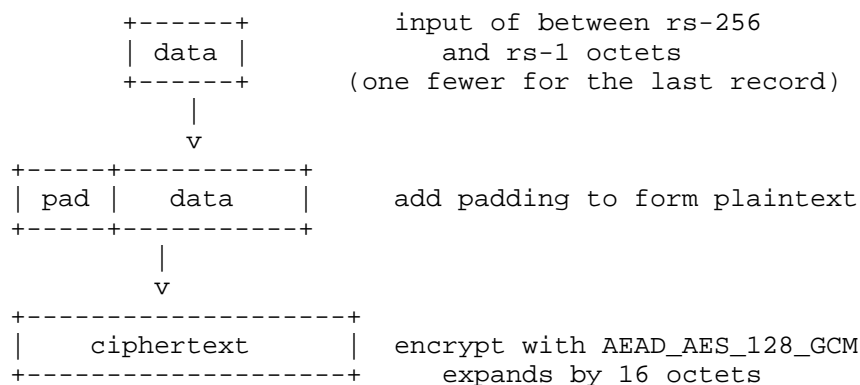
The "aesgcm128" HTTP content-coding indicates that a payload has been encrypted using Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) as identified as AEAD_AES_128_GCM in [RFC5116], Section 5.1. The AEAD_AES_128_GCM algorithm uses a 128 bit content encryption key.

When this content-coding is in use, the Encryption header field (Section 3) describes how encryption has been applied. The

Encryption-Key header field (Section 4) can be included to describe how the content encryption key is derived or retrieved.

The "aesgcm128" content-coding uses a single fixed set of encryption primitives. Cipher suite agility is achieved by defining a new content-coding scheme. This ensures that only the HTTP Accept-Encoding header field is necessary to negotiate the use of encryption.

The "aesgcm128" content-coding uses a fixed record size. The resulting encoding is a series of fixed-size records, with a final record that is one or more octets shorter than a fixed sized record.



The record size determines the length of each portion of plaintext that is enciphered, with the exception of the final record, which is necessarily smaller. The record size defaults to 4096 octets, but can be changed using the "rs" parameter on the Encryption header field.

AEAD_AES_128_GCM expands ciphertext to be 16 octets longer than its input plaintext. Therefore, the length of each enciphered record other than the last is equal to the value of the "rs" parameter plus 16 octets. A receiver MUST fail to decrypt if the final record ciphertext is 16 octets or less in size. Valid records always contain at least one byte of padding and a 16 octet authentication tag.

Each record contains between 1 and 256 octets of padding, inserted into a record before the enciphered content. Padding consists of a length byte, followed that number of zero-valued octets. A receiver MUST fail to decrypt if any padding octet other than the first is non-zero, or a record has more padding than the record size can accommodate.

The nonce for each record is a 96-bit value constructed from the record sequence number and the input keying material. Nonce derivation is covered in Section 3.3.

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

A sequence of full-sized records can be truncated to produce a shorter sequence of records with valid authentication tags. To prevent an attacker from truncating a stream, an encoder **MUST** append a record that contains only padding and is smaller than the full record size if the final record ends on a record boundary. A receiver **MUST** treat the stream as failed due to truncation if the final record is the full record size.

A consequence of this record structure is that range requests [RFC7233] and random access to encrypted payload bodies are possible at the granularity of the record size. However, without data from adjacent ranges, partial records cannot be used. Thus, it is best if records start and end on multiples of the record size, plus the 16 octet authentication tag size.

3. The Encryption HTTP Header Field

The "Encryption" HTTP header field describes the encrypted content encoding(s) that have been applied to a payload body, and therefore how those content encoding(s) can be removed.

The "Encryption" header field uses the extended ABNF syntax defined in Section 1.2 of [RFC7230] and the "parameter" rule from [RFC7231]

```
Encryption-val = #encryption_params  
encryption_params = [ parameter *( ";" parameter ) ]
```

If the payload is encrypted more than once (as reflected by having multiple content-codings that imply encryption), each application of the content encoding is reflected in the Encryption header field, in the order in which they were applied.

The Encryption header **MAY** be omitted if the sender does not intend for the immediate recipient to be able to decrypt the payload body. Alternatively, the Encryption header field **MAY** be omitted if the sender intends for the recipient to acquire the header field by other means.

Servers processing PUT requests **MUST** persist the value of the Encryption header field, unless they remove the content-coding by decrypting the payload.

3.1. Encryption Header Field Parameters

The following parameters are used in determining the content encryption key that is used for encryption:

keyid: The "keyid" parameter contains a string that identifies the keying material that is used. The "keyid" parameter SHOULD be included, unless key identification is guaranteed by other means. The "keyid" parameter MUST be used if keying material included in an Encryption-Key header field is needed to derive the content encryption key.

salt: The "salt" parameter contains a base64 URL-encoded octets that is used as salt in deriving a unique content encryption key (see Section 3.2). The "salt" parameter MUST be present, and MUST be exactly 16 octets long when decoded. The "salt" parameter MUST NOT be reused for two different payload bodies that have the same input keying material; generating a random salt for every application of the content encoding ensures that content encryption key reuse is highly unlikely.

rs: The "rs" parameter contains a positive decimal integer that describes the record size in octets. This value MUST be greater than 1. If the "rs" parameter is absent, the record size defaults to 4096 octets.

3.2. Content Encryption Key Derivation

In order to allow the reuse of keying material for multiple different HTTP messages, a content encryption key is derived for each message. The content encryption key is derived from the decoded value of the "salt" parameter using the HMAC-based key derivation function (HKDF) described in [RFC5869] using the SHA-256 hash algorithm [FIPS180-2].

The decoded value of the "salt" parameter is the salt input to HKDF function. The keying material identified by the "keyid" parameter is the input keying material (IKM) to HKDF. Input keying material can either be prearranged, or can be described using the Encryption-Key header field (Section 4). The first step of HKDF is therefore:

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

AEAD_AES_128_GCM requires a 16 octet (128 bit) content encryption key, so the length (L) parameter to HKDF is 16. The info parameter is set to the ASCII-encoded string "Content-Encoding: aesgcm128". The second step of HKDF can therefore be simplified to the first 16 octets of a single HMAC:

```
CEK = HMAC-SHA-256(PRK, "Content-Encoding: aesgcm128" || 0x01)
```

3.3. Nonce Derivation

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

The input keying material and salt values are input to HKDF with different info and length parameters. The info parameter for the nonce is the ASCII-encoded string "Content-Encoding: nonce" and the length (L) parameter is 12 octets.

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

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

```
NONCE = HMAC-SHA-256(PRK, "Content-Encoding: nonce" || 0x01) ^ SEQ
```

4. Encryption-Key Header Field

An Encryption-Key header field can be used to describe the input keying material used in the Encryption header field.

The Encryption-Key header field uses the extended ABNF syntax defined in Section 1.2 of [RFC7230] and the "parameter" rule from [RFC7231].

```
Encryption-Key-val = #encryption_key_params  
encryption_key_params = [ parameter *( ";" parameter ) ]
```

keyid: The "keyid" parameter corresponds to the "keyid" parameter in the Encryption header field.

aesgcm128: The "aesgcm128" parameter contains the URL-safe base64 [RFC4648] octets of the input keying material.

dh: The "dh" parameter contains an ephemeral Diffie-Hellman share. This form of the header field can be used to encrypt content for a specific recipient.

The input keying material used by the content-encoding key derivation (see Section 3.2) can be determined based on the information in the Encryption-Key header field. The method for key derivation depends on the parameters that are present in the header field.

The value or values provided in the Encryption-Key header field is valid only for the current HTTP message unless additional information indicates a greater scope.

Note that different methods for determining input keying material will produce different amounts of data. The HKDF process ensures that the final content encryption key is the necessary size.

Alternative methods for determining input keying material MAY be defined by specifications that use this content-encoding.

4.1. Explicit Key

The "aesgcm128" parameter is decoded and used as the input keying material for the "aesgcm128" content encoding. The "aesgcm128" parameter MUST decode to at least 16 octets in order to be used as input keying material for "aesgcm128" content encoding.

Other key determination parameters can be ignored if the "aesgcm128" parameter is present.

4.2. Diffie-Hellman

The "dh" parameter is included to describe a Diffie-Hellman share, either modp (or finite field) Diffie-Hellman [DH] or elliptic curve Diffie-Hellman (ECDH) [RFC4492].

This share is combined with other information at the recipient to determine the HKDF input keying material. In order for the exchange to be successful, the following information MUST be established out of band:

- o Which Diffie-Hellman form is used.
- o The modp group or elliptic curve that will be used.
- o The format of the ephemeral public share that is included in the "dh" parameter. For instance, using ECDH both parties need to agree whether this is an uncompressed or compressed point.

In addition to identifying which content-encoding this input keying material is used for, the "keyid" parameter is used to identify this additional information at the receiver.

The intended recipient recovers their private key and are then able to generate a shared secret using the appropriate Diffie-Hellman process.

Specifications that rely on an Diffie-Hellman exchange for determining input keying material MUST either specify the parameters for Diffie-Hellman (group parameters, or curves and point format) that are used, or describe how those parameters are negotiated between sender and receiver.

5. Examples

5.1. Successful GET Response

```
HTTP/1.1 200 OK
Content-Type: application/octet-stream
Content-Encoding: aesgcm128
Connection: close
Encryption: keyid="http://example.org/bob/keys/123";
           salt="XZwpw6o37R-6qoZjw6KwAw"
```

[encrypted payload]

Here, a successful HTTP GET response has been encrypted using input keying material that is identified by a URI.

Note that the media type has been changed to "application/octet-stream" to avoid exposing information about the content.

5.2. Encryption and Compression

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Encoding: aesgcm128, gzip
Transfer-Encoding: chunked
Encryption: keyid="mailto:me@example.com";
           salt="m2hJ_NttRtFyUiMRPwfpHA"
```

[encrypted payload]

5.3. Encryption with More Than One Key


```
PUT /thing HTTP/1.1
Host: storage.example.com
Content-Type: application/http
Content-Encoding: aesgcm128, aesgcm128
Content-Length: 1234
Encryption: keyid="mailto:me@example.com";
            salt="NfzOeuV5USPRA-n_9s1Lag",
            keyid="http://example.org/bob/keys/123";
            salt="bDMSGoc2uobK_IhavSHsHA"; rs=1200
```

[encrypted payload]

Here, a PUT request has been encrypted twice with different input keying material; decrypting twice is necessary to read the content. The outer layer of encryption uses a 1200 octet record size.

5.4. Encryption with Explicit Key

```
HTTP/1.1 200 OK
Content-Length: 32
Content-Encoding: aesgcm128
Encryption: keyid="a1"; salt="ibZx1RNz537h1XNkRcPpJA"
Encryption-Key: keyid="a1"; aesgcm128="9Z57YCb3dK95dSsdFJbkag"
```

zK3kpG__Z8whjIkG6RYgPz11oUkTKcxPy9WP-VPMfuc

This example shows the string "I am the walrus" encrypted using an directly provided value for the input keying material. The content body contains a single record only and is shown here encoded in URL-safe base64 for presentation reasons only.

5.5. Diffie-Hellman Encryption

```
HTTP/1.1 200 OK
Content-Length: 32
Content-Encoding: aesgcm128
Encryption: keyid="dhkey"; salt="5hpuYfxDzG6nSs9-EQuaBg"
Encryption-Key: keyid="dhkey";
                dh="BLsyIPbDn6bquEOwHaju2gj8kUVof1zTtPs_6fGoock_
                dwxilBcgFtObPVnic4alcEucx8I6G8HmEZCJnAl36Zg"
```

BmuHqRzdD4WlmibxglrPiRHZRSY49Dzdm6jHrWXzZrE

This example shows the same string, "I am the walrus", encrypted using ECDH over the P-256 curve [FIPS186]. The content body is shown here encoded in URL-safe base64 for presentation reasons only.

The receiver (in this case, the HTTP client) uses a key pair that is identified by the string "dhkey" and the sender (the server) uses a key pair for which the public share is included in the "dh" parameter above. The keys shown below use uncompressed points [X.692] encoded using URL-safe base64. Line wrapping is added for presentation purposes only.

Receiver:

```
private key: iCjNf8v4ox_glrJuSs_gbNmYuUYx76ZRruQs_CHRzDg
public key: BPM1w41cSD4BMeBTY0Fz9ryLM-LeM22Dvt0gaLRukf05
           rMhzFAvxVW_mipg5O0hkWad9ZWW0uMRO2Nrd32v8odQ
```

Sender:

```
private key: W0cxgeHDZkR3uMQYAbVgF5swKQUAR7DgoTaaQVlA-Fg
public key: <the value of the "dh" parameter>
```

6. Security Considerations

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

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

6.1. Key and Nonce Reuse

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

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

6.2. Content Integrity

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

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

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

6.3. Leaking Information in Headers

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

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

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

1. Determine that it is not an issue. For example, if it is expected that all content stored will be "application/json", or another very common media type, exposing the Content-Type header field could be an acceptable risk.
2. If it is considered sensitive information and it is possible to determine it through other means (e.g., out of band, using hints in other representations, etc.), omit the relevant headers, and/or normalize them. In the case of Content-Type, this could be accomplished by always sending Content-Type: application/octet-stream (the most generic media type), or no Content-Type at all.
3. If it is considered sensitive information and it is not possible to convey it elsewhere, encapsulate the HTTP message using the application/http media type (Section 8.3.2 of [RFC7230]), encrypting that as the payload of the "outer" message.

6.4. Poisoning Storage

This mechanism only offers encryption of content; it does not perform authentication or authorization, which still needs to be performed (e.g., by HTTP authentication [RFC7235]).

This is especially relevant when a HTTP PUT request is accepted by a server; if the request is unauthenticated, it becomes possible for a third party to deny service and/or poison the store.

6.5. Sizing and Timing Attacks

Applications using this mechanism need to be aware that the size of encrypted messages, as well as their timing, HTTP methods, URIs and so on, may leak sensitive information.

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

7. IANA Considerations

7.1. The "aesgcm128" HTTP Content Encoding

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

- o Name: aesgcm128
- o Description: AES-GCM encryption with a 128-bit content encryption key
- o Reference: this specification

7.2. Encryption Header Fields

This memo registers the "Encryption" HTTP header field in the Permanent Message Header Registry, as detailed in Section 3.

- o Field name: Encryption
- o Protocol: HTTP
- o Status: Standard
- o Reference: this specification

- o Notes:

This memo registers the "Encryption-Key" HTTP header field in the Permanent Message Header Registry, as detailed in Section 4.

- o Field name: Encryption-Key
- o Protocol: HTTP
- o Status: Standard
- o Reference: this specification
- o Notes:

7.3. The HTTP Encryption Parameter Registry

This memo establishes a registry for parameters used by the "Encryption" header field under the "Hypertext Transfer Protocol (HTTP) Parameters" grouping. The "Hypertext Transfer Protocol (HTTP) Encryption Parameters" operates under an "Specification Required" policy [RFC5226].

Entries in this registry are expected to include the following information:

- o Parameter Name: The name of the parameter.
- o Purpose: A brief description of the purpose of the parameter.
- o Reference: A reference to a specification that defines the semantics of the parameter.

The initial contents of this registry are:

7.3.1. keyid

- o Parameter Name: keyid
- o Purpose: Identify the key that is in use.
- o Reference: this document

7.3.2. salt

- o Parameter Name: salt

- o Purpose: Provide a source of entropy for derivation of a content encryption key. This value is mandatory.

- o Reference: this document

7.3.3. rs

- o Parameter Name: rs
- o Purpose: The size of the encrypted records.
- o Reference: this document

7.4. The HTTP Encryption-Key Parameter Registry

This memo establishes a registry for parameters used by the "Encryption-Key" header field under the "Hypertext Transfer Protocol (HTTP) Parameters" grouping. The "Hypertext Transfer Protocol (HTTP) Encryption Parameters" operates under an "Specification Required" policy [RFC5226].

Entries in this registry are expected to include the following information:

- o Parameter Name: The name of the parameter.
- o Purpose: A brief description of the purpose of the parameter.
- o Reference: A reference to a specification that defines the semantics of the parameter.

The initial contents of this registry are:

7.4.1. keyid

- o Parameter Name: keyid
- o Purpose: Identify the key that is in use.
- o Reference: this document

7.4.2. aesgcm128

- o Parameter Name: aesgcm128
- o Purpose: Provide an explicit input keying material value for the aesgcm128 content encoding.

- o Reference: this document

7.4.3. dh

- o Parameter Name: dh
- o Purpose: Carry a modp or elliptic curve Diffie-Hellman share used to derive input keying material.
- o Reference: this document

8. References

8.1. Normative References

- [DH] Diffie, W. and M. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, V.IT-22 n.6 , June 1977.
- [FIPS180-2] Department of Commerce, National., "NIST FIPS 180-2, Secure Hash Standard", August 2002.
- [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>>.
- [RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <<http://www.rfc-editor.org/info/rfc4492>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<http://www.rfc-editor.org/info/rfc4648>>.
- [RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <<http://www.rfc-editor.org/info/rfc5116>>.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<http://www.rfc-editor.org/info/rfc5869>>.

- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<http://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<http://www.rfc-editor.org/info/rfc7231>>.

8.2. Informative References

- [FIPS186] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", NIST PUB 186-4 , July 2013.
- [RFC4880] Callas, J., Donnerhacke, L., Finney, H., Shaw, D., and R. Thayer, "OpenPGP Message Format", RFC 4880, DOI 10.17487/RFC4880, November 2007, <<http://www.rfc-editor.org/info/rfc4880>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <<http://www.rfc-editor.org/info/rfc5652>>.
- [RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <<http://www.rfc-editor.org/info/rfc7233>>.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <<http://www.rfc-editor.org/info/rfc7235>>.
- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <<http://www.rfc-editor.org/info/rfc7516>>.

- [RFC7540] Belshé, 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>>.
- [X.692] ANSI, "Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62 , 1998.
- [XMLENC] Eastlake, D., Reagle, J., Imamura, T., Dillaway, B., and E. Simon, "XML Encryption Syntax and Processing", W3C REC , December 2002, <<http://www.w3.org/TR/xmlenc-core/>>.

Appendix A. JWE Mapping

The "aescm128" content encoding can be considered as a sequence of JSON Web Encryption (JWE) objects [RFC7516], each corresponding to a single fixed size record. The following transformations are applied to a JWE object that might be expressed using the JWE Compact Serialization:

- o The JWE Protected Header is fixed to a value { "alg": "dir", "enc": "A128GCM" }, describing direct encryption using AES-GCM with a 128-bit content encryption key. This header is not transmitted, it is instead implied by the value of the Content-Encoding header field.
- o The JWE Encrypted Key is empty, as stipulated by the direct encryption algorithm.
- o The JWE Initialization Vector ("iv") for each record is set to the exclusive or of the 96-bit record sequence number, starting at zero, and a value derived from the input keying material (see Section 3.3). This value is also not transmitted.
- o The final value is the concatenated JWE Ciphertext and the JWE Authentication Tag, both expressed without URL-safe Base 64 encoding. The "." separator is omitted, since the length of these fields is known.

Thus, the example in Section 5.4 can be rendered using the JWE Compact Serialization as:

```
eyJYXnIjogImRpciIsICJlbmMiOiAiQTEyOEdDTSIgZQ..AAAAAAAAAAAAAAAAA.  
LwTC-fwdKh8de0smD2jfzA.ehlvURhu65M2lxhctbbntA
```

Where the first line represents the fixed JWE Protected Header, JWE Encrypted Key, and JWE Initialization Vector, all of which are

determined algorithmically. The second line contains the encoded body, split into JWE Ciphertext and JWE Authentication Tag.

Appendix B. Acknowledgements

Mark Nottingham was an original author of this document.

The following people provided valuable input: Richard Barnes, David Benjamin, Peter Beverloo, Mike Jones, Stephen Farrell, Adam Langley, John Mattsson, Eric Rescorla, and Jim Schaad.

Author's Address

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

HTTPbis
Internet-Draft
Updates: 6265 (if approved)
Intended status: Standards Track
Expires: June 2, 2016

M. West
Google, Inc
November 30, 2015

Cookie Prefixes
draft-west-cookie-prefixes-05

Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 2, 2016.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

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

1. Introduction

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

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

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

2. Terminology and notation

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

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

3. Prefixes

3.1. The "__Secure-" prefix

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

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

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

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

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

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

3.2. The "__Host-" prefix

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

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

The following cookies would always be rejected:

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

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

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

4. User Agent Requirements

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

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

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

5. Aesthetic Considerations

5.1. Not pretty.

Prefixes are ugly. :(

5.2. Why "__"?

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

6. Security Considerations

6.1. Secure Origins Only

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

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

6.2. Limitations

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

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.

7.2. Informative References

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

[Lawrence2015]

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

[POWERFUL-FEATURES]

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

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

[SECURE-CONTEXTS]

West, M., "Secure Contexts", n.d., <<https://w3c.github.io/webappsec-secure-contexts/>>.

Appendix A. Acknowledgements

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

Author's Address

Mike West
Google, Inc

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

HTTPbis
Internet-Draft
Updates: 6265 (if approved)
Intended status: Standards Track
Expires: October 8, 2016

M. West
Google, Inc
M. Goodwin
Mozilla
April 6, 2016

Same-site Cookies
draft-west-first-party-cookies-07

Abstract

This document updates RFC6265 by defining a "SameSite" attribute which allows servers to assert that a cookie ought not to be sent along with cross-site requests. This assertion allows user agents to mitigate the risk of cross-origin information leakage, and provides some protection against cross-site request forgery attacks.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 8, 2016.

Copyright Notice

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

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

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

Table of Contents

1. Introduction	2
1.1. Goals	3
1.2. Examples	3
2. Terminology and notation	4
2.1. "Same-site" and "cross-site" Requests	4
2.1.1. Document-based requests	5
2.1.2. Worker-based requests	6
3. Server Requirements	7
3.1. Grammar	7
3.2. Semantics of the "SameSite" Attribute (Non-Normative) . .	8
4. User Agent Requirements	8
4.1. The "SameSite" attribute	8
4.1.1. "Strict" and "Lax" enforcement	8
4.2. Monkey-patching the Storage Model	9
4.3. Monkey-patching the "Cookie" header	10
5. Authoring Considerations	10
5.1. Defense in depth	10
5.2. Top-level Navigations	11
5.3. Mashups and Widgets	11
6. Privacy Considerations	11
6.1. Server-controlled	11
6.2. Pervasive Monitoring	12
7. References	12
7.1. Normative References	12
7.2. Informative References	13
Appendix A. Acknowledgements	14
Authors' Addresses	14

1. Introduction

Section 8.2 of [RFC6265] eloquently notes that cookies are a form of ambient authority, attached by default to requests the user agent sends on a user's behalf. Even when an attacker doesn't know the contents of a user's cookies, she can still execute commands on the user's behalf (and with the user's authority) by asking the user agent to send HTTP requests to unwary servers.

Here, we update [RFC6265] with a simple mitigation strategy that allows servers to declare certain cookies as "same-site", meaning they should not be attached to "cross-site" requests (as defined in section 2.1).

Note that the mechanism outlined here is backwards compatible with the existing cookie syntax. Servers may serve these cookies to all user agents; those that do not support the "SameSite" attribute will simply store a cookie which is attached to all relevant requests, just as they do today.

1.1. Goals

These cookies are intended to provide a solid layer of defense-in-depth against attacks which require embedding an authenticated request into an attacker-controlled context:

1. Timing attacks which yield cross-origin information leakage (such as those detailed in [pixel-perfect]) can be substantially mitigated by setting the "SameSite" attribute on authentication cookies. The attacker will only be able to embed unauthenticated resources, as embedding mechanisms such as "<iframe>" will yield cross-site requests.
2. Cross-site script inclusion (XSSI) attacks are likewise mitigated by setting the "SameSite" attribute on authentication cookies. The attacker will not be able to include authenticated resources via "<script>" or "<link>", as these embedding mechanisms will likewise yield cross-site requests.
3. Cross-site request forgery (CSRF) attacks which rely on top-level navigation (HTML "<form>" POSTs, for instance) can also be mitigated by treating these navigational requests as "cross-site".
4. Same-site cookies have some marginal value for policy or regulatory purposes, as cookies which are not delivered with cross-site requests cannot be directly used for tracking purposes. It may be valuable for an origin to assert that its cookies should not be sent along with cross-site requests in order to limit its exposure to non-technical risk.

1.2. Examples

Same-site cookies are set via the "SameSite" attribute in the "Set-Cookie" header field. That is, given a server's response to a user agent which contains the following header field:

```
Set-Cookie: SID=31d4d96e407aad42; SameSite=Strict
```

Subsequent requests from that user agent can be expected to contain the following header field if and only if both the requested resource and the resource in the top-level browsing context match the cookie.

2. Terminology and notation

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

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

Two sequences of octets are said to case-insensitively match each other if and only if they are equivalent under the "i;ascii-casemap" collation defined in [RFC4790].

The terms "active document", "ancestor browsing context", "browsing context", "document", "WorkerGlobalScope", "sandboxed origin browsing context flag", "parent browsing context", "the worker's Documents", "nested browsing context", and "top-level browsing context" are defined in [HTML].

"Service Workers" are defined in the Service Workers specification [SERVICE-WORKERS].

The term "origin", the mechanism of deriving an origin from a URI, and the "the same" matching algorithm for origins are defined in [RFC6454].

"Safe" HTTP methods include "GET", "HEAD", "OPTIONS", and "TRACE", as defined in Section 4.2.1 of [RFC7231].

The term "public suffix" is defined in a note in Section 5.3 of [RFC6265] as "a domain that is controlled by a public registry". For example, "example.com"'s public suffix is "com". User agents SHOULD use an up-to-date public suffix list, such as the one maintained by Mozilla at [PSL].

An origin's "registrable domain" is the origin's host's public suffix plus the label to its left. That is, "https://www.example.com"'s registrable domain is "example.com". This concept is defined more rigorously in [PSL].

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

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

A request is "same-site" if its target's URI's origin's registrable domain is an exact match for the request's initiator's "site for cookies", and "cross-site" otherwise. To be more precise, for a

given request ("request"), the following algorithm returns "same-site" or "cross-site":

1. If "request"'s client is "null", return "same-site".
2. Let "site" be "request"'s client's "site for cookies" (as defined in the following sections).
3. Let "target" be the registrable domain of "request"'s current url.
4. If "site" is an exact match for "target", return "same-site".
5. Return "cross-site".

2.1.1.1. Document-based requests

The URI displayed in a user agent's address bar is the only security context directly exposed to users, and therefore the only signal users can reasonably rely upon to determine whether or not they trust a particular website. The registrable domain of that URI's origin represents the context in which a user most likely believes themselves to be interacting. We'll label this domain the "top-level site".

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

For documents which are displayed in nested browsing contexts, we need to audit the origins of each of a document's ancestor browsing contexts' active documents in order to account for the "multiple-nested scenarios" described in Section 4 of [RFC7034]. These document's "site for cookies" is the top-level site if and only if the document and each of its ancestor documents' origins have the same registrable domain as the top-level site. Otherwise its "site for cookies" is the empty string.

Given a Document ("document"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Let "top-document" be the active document in "document"'s browsing context's top-level browsing context.
2. Let "top-origin" be the origin of "top-document"'s URI if "top-document"'s sandboxed origin browsing context flag is set, and "top-document"'s origin otherwise.

3. Let "documents" be a list containing "document" and each of "document"'s ancestor browsing contexts' active documents.
4. For each "item" in "documents":
 1. Let "origin" be the origin of "item"'s URI if "item"'s sandboxed origin browsing context flag is set, and "item"'s origin otherwise.
 2. If "origin"'s host's registrable domain is not an exact match for "top-origin"'s host's registrable domain, return the empty string.
5. Return "top-site".

2.1.1.2. Worker-based requests

Worker-driven requests aren't as clear-cut as document-driven requests, as there isn't a clear link between a top-level browsing context and a worker. This is especially true for Service Workers [SERVICE-WORKERS], which may execute code in the background, without any document visible at all.

Note: The descriptions below assume that workers must be same-origin with the documents that instantiate them. If this invariant changes, we'll need to take the worker's script's URI into account when determining their status.

2.1.1.2.1. Dedicated and Shared Workers

Dedicated workers are simple, as each dedicated worker is bound to one and only one document. Requests generated from a dedicated worker (via "importScripts", "XMLHttpRequest", "fetch()", etc) define their "site for cookies" as that document's "site for cookies".

Shared workers may be bound to multiple documents at once. As it is quite possible for those documents to have distinct "site for cookie" values, the worker's "site for cookies" will be the empty string in cases where the values diverge, and the shared value in cases where the values agree.

Given a WorkerGlobalScope ("worker"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Let "site" be "worker"'s origin's host's registrable domain.
2. For each "document" in "worker"'s Documents:

1. Let "document-site" be "document"'s "site for cookies" (as defined in Section 2.1.1).
2. If "document-site" is not an exact match for "site", return the empty string.
3. Return "site".

2.1.2.2. Service Workers

Service Workers are more complicated, as they act as a completely separate execution context with only tangential relationship to the Document which registered them.

Requests which simply pass through a service worker will be handled as described above: the request's client will be the Document or Worker which initiated the request, and its "site for cookies" will be those defined in Section 2.1.1 and Section 2.1.2.1

Requests which are initiated by the Service Worker itself (via a direct call to "fetch()", for instance), on the other hand, will have a client which is a ServiceWorkerGlobalScope. Its "site for cookies" will be the registrable domain of the Service Worker's URI.

Given a ServiceWorkerGlobalScope ("worker"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Return "worker"'s origin's host's registrable domain.

3. Server Requirements

This section describes extensions to [RFC6265] necessary to implement the server-side requirements of the "SameSite" attribute.

3.1. Grammar

Add "SameSite" to the list of accepted attributes in the "Set-Cookie" header field's value by replacing the "cookie-av" token definition in Section 4.1.1 of [RFC6265] with the following ABNF grammar:

```
cookie-av      = expires-av / max-age-av / domain-av /  
                path-av / secure-av / httponly-av /  
                samesite-av / extension-av  
samesite-av    = "SameSite" / "SameSite=" samesite-value  
samesite-value = "Strict" / "Lax"
```

3.2. Semantics of the "SameSite" Attribute (Non-Normative)

The "SameSite" attribute limits the scope of the cookie such that it will only be attached to requests if those requests are "same-site", as defined by the algorithm in Section 2.1. For example, requests for "https://example.com/sekrit-image" will attach same-site cookies if and only if initiated from a context whose "site for cookies" is "example.com".

If the "SameSite" attribute's value is "Strict", or if the value is invalid, the cookie will only be sent along with "same-site" requests. If the value is "Lax", the cookie will be sent with "same-site" requests, and with "cross-site" top-level navigations, as described in Section 4.1.1.

The changes to the "Cookie" header field suggested in Section 4.3 provide additional detail.

4. User Agent Requirements

This section describes extensions to [RFC6265] necessary in order to implement the client-side requirements of the "SameSite" attribute.

4.1. The "SameSite" attribute

The following attribute definition should be considered part of the the "Set-Cookie" algorithm as described in Section 5.2 of [RFC6265]:

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

1. If "cookie-av"'s "attribute-value" is not a case-sensitive match for "Strict" or "Lax", ignore the "cookie-av".
2. Let "enforcement" be "Lax" if "cookie-av"'s "attribute-value" is a case-insensitive match for "Lax", and "Strict" otherwise.
3. Append an attribute to the "cookie-attribute-list" with an "attribute-name" of "SameSite" and an "attribute-value" of "enforcement".

4.1.1. "Strict" and "Lax" enforcement

By default, same-site cookies will not be sent along with top-level navigations. As discussed in Section 5.2, this might or might not be compatible with existing session management systems. In the interests of providing a drop-in mechanism that mitigates the risk of CSRF attacks, developers may set the "SameSite" attribute in a "Lax"

enforcement mode that carves out an exception which sends same-site cookies along with cross-site requests if and only if they are top-level navigations which use a "safe" (in the [RFC7231] sense) HTTP method.

Lax enforcement provides reasonable defense in depth against CSRF attacks that rely on unsafe HTTP methods (like "POST"), but do not offer a robust defense against CSRF as a general category of attack:

1. Attackers can still pop up new windows or trigger top-level navigations in order to create a "same-site" request (as described in section 2.1), which is only a speedbump along the road to exploitation.
2. Features like "<link rel='prerender'>" [prerendering] can be exploited to create "same-site" requests without the risk of user detection.

When possible, developers should use a session management mechanism such as that described in Section 5.2 to mitigate the risk of CSRF more completely.

4.2. Monkey-patching the Storage Model

Note: There's got to be a better way to specify this. Until I figure out what that is, monkey-patching!

Alter Section 5.3 of [RFC6265] as follows:

1. Add "samesite-flag" to the list of fields stored for each cookie. This field's value is one of "None", "Strict", or "Lax".
2. Before step 11 of the current algorithm, add the following:
 1. If the "cookie-attribute-list" contains an attribute with an "attribute-name" of "SameSite", set the cookie's "samesite-flag" to "attribute-value" ("Strict" or "Lax"). Otherwise, set the cookie's "samesite-flag" to "None".
 2. If the cookie's "samesite-flag" is not "None", and the request which generated the cookie's client's "site for cookies" is not an exact match for "request-uri"'s host's registrable domain, then abort these steps and ignore the newly created cookie entirely.

4.3. Monkey-patching the "Cookie" header

Note: There's got to be a better way to specify this. Until I figure out what that is, monkey-patching!

Alter Section 5.4 of [RFC6265] as follows:

1. Add the following requirement to the list in step 1:
 - * If the cookie's "samesite-flag" is not "None", and the HTTP request is cross-site (as defined in Section 2.1 then exclude the cookie unless all of the following statements hold:
 1. "samesite-flag" is "Lax"
 2. The HTTP request's method is "safe".
 3. The HTTP request's target browsing context is a top-level browsing context.

Note that the modifications suggested here concern themselves only with the "site for cookies" of the request's client, and the registrable domain of the resource being requested. The cookie's "domain", "path", and "secure" attributes do not come into play for these comparisons.

5. Authoring Considerations

5.1. Defense in depth

"SameSite" cookies offer a robust defense against CSRF attack when deployed in strict mode, and when supported by the client. It is, however, prudent to ensure that this designation is not the extent of a site's defense against CSRF, as same-site navigations and submissions can certainly be executed in conjunction with other attack vectors such as cross-site scripting.

Developers are strongly encouraged to deploy the usual server-side defenses (CSRF tokens, ensuring that "safe" HTTP methods are idempotent, etc) to mitigate the risk more fully.

Additionally, client-side techniques such as those described in [app-isolation] may also prove effective against CSRF, and are certainly worth exploring in combination with "SameSite" cookies.

5.2. Top-level Navigations

Setting the "SameSite" attribute in "strict" mode provides robust defense in depth against CSRF attacks, but has the potential to confuse users unless sites' developers carefully ensure that their session management systems deal reasonably well with top-level navigations.

Consider the scenario in which a user reads their email at MegaCorp Inc's webmail provider "https://example.com/". They might expect that clicking on an emailed link to "https://projects.com/secret/project" would show them the secret project that they're authorized to see, but if "projects.com" has marked their session cookies as "SameSite", then this cross-site navigation won't send them along with the request. "projects.com" will render a 404 error to avoid leaking secret information, and the user will be quite confused.

Developers can avoid this confusion by adopting a session management system that relies on not one, but two cookies: one conceptually granting "read" access, another granting "write" access. The latter could be marked as "SameSite", and its absence would provide a reauthentication step before executing any non-idempotent action. The former could drop the "SameSite" attribute entirely, or choose the "Lax" version of enforcement, in order to allow users access to data via top-level navigation.

5.3. Mashups and Widgets

The "SameSite" attribute is inappropriate for some important use-cases. In particular, note that content intended for embedding in a cross-site contexts (social networking widgets or commenting services, for instance) will not have access to such cookies. Cross-site cookies may be required in order to provide seamless functionality that relies on a user's state.

Likewise, some forms of Single-Sign-On might require authentication in a cross-site context; these mechanisms will not function as intended with same-site cookies.

6. Privacy Considerations

6.1. Server-controlled

Same-site cookies in and of themselves don't do anything to address the general privacy concerns outlined in Section 7.1 of [RFC6265]. The attribute is set by the server, and serves to mitigate the risk of certain kinds of attacks that the server is worried about. The user is not involved in this decision. Moreover, a number of side-

channels exist which could allow a server to link distinct requests even in the absence of cookies. Connection and/or socket pooling, Token Binding, and Channel ID all offer explicit methods of identification that servers could take advantage of.

6.2. Pervasive Monitoring

As outlined in [RFC7258], pervasive monitoring is an attack. Cookies play a large part in enabling such monitoring, as they are responsible for maintaining state in HTTP connections. We considered restricting same-site cookies to secure contexts [secure-contexts] as a mitigation but decided against doing so, as this feature should result in a strict reduction in the number of cookies floating around in cross-site contexts. That is, even if "http://not-example.com" embeds a resource from "http://example.com/", that resource will not be "same-site", and "http://example.com"'s cookies simply cannot be used to correlate user behavior across distinct origins.

7. References

7.1. Normative References

- [FETCH] van Kesteren, A., "Fetch", n.d., <<https://fetch.spec.whatwg.org/>>.
- [HTML] Hickson, I., Pieters, S., van Kesteren, A., Jaegenstedt, P., and D. Denicola, "HTML", n.d., <<https://html.spec.whatwg.org/>>.
- [PSL] "Public Suffix List", n.d., <<https://publicsuffix.org/list/>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, DOI 10.17487/RFC4790, March 2007, <<http://www.rfc-editor.org/info/rfc4790>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<http://www.rfc-editor.org/info/rfc5234>>.

- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<http://www.rfc-editor.org/info/rfc6265>>.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <<http://www.rfc-editor.org/info/rfc6454>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<http://www.rfc-editor.org/info/rfc7231>>.
- [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <<http://www.rfc-editor.org/info/rfc7258>>.
- [SERVICE-WORKERS]
Russell, A., Song, J., and J. Archibald, "Service Workers", n.d., <<http://www.w3.org/TR/service-workers/>>.

7.2. Informative References

- [app-isolation]
Chen, E., Bau, J., Reis, C., Barth, A., and C. Jackson, "App Isolation - Get the Security of Multiple Browsers with Just One", n.d., <<http://www.collinjackson.com/research/papers/appisolation.pdf>>.
- [pixel-perfect]
Stone, P., "Pixel Perfect Timing Attacks with HTML5", n.d., <http://www.contextis.com/documents/2/Browser_Timing_Attacks.pdf>.
- [prerendering]
Bentzel, C., "Chrome Prerendering", n.d., <<https://www.chromium.org/developers/design-documents/prerender>>.
- [RFC7034] Ross, D. and T. Gondrom, "HTTP Header Field X-Frame-Options", RFC 7034, DOI 10.17487/RFC7034, October 2013, <<http://www.rfc-editor.org/info/rfc7034>>.
- [samedomain-cookies]
Goodwin, M. and J. Walker, "SameDomain Cookie Flag", 2011, <<http://people.mozilla.org/~mgoodwin/SameDomain/samedomain-latest.txt>>.

[secure-contexts]

West, M., "Secure Contexts", n.d., <<https://w3c.github.io/webappsec-secure-contexts/>>.

Appendix A. Acknowledgements

The same-site cookie concept documented here is indebted to Mark Goodwin's and Joe Walker's [samedomain-cookies]. Michal Zalewski, Artur Janc, Ryan Sleevi, and Adam Barth provided particularly valuable feedback on this document.

Authors' Addresses

Mike West
Google, Inc

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

Mark Goodwin
Mozilla

Email: mgoodwin@mozilla.com
URI: <https://www.computerist.org/>

HTTPbis
Internet-Draft
Updates: 6265 (if approved)
Intended status: Standards Track
Expires: July 10, 2016

M. West
Google, Inc
January 7, 2016

Deprecate modification of 'secure' cookies from non-secure origins
draft-west-leave-secure-cookies-alone-05

Abstract

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

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 10, 2016.

Copyright Notice

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

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

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

Table of Contents

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

1. Introduction

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

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

2. Terminology and notation

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

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

3. Recommendations

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

1. After step 8 of the current algorithm, which sets the cookie's "secure-only-flag", execute the following step:
 1. If the "scheme" component of the "request-uri" does not denote a "secure" protocol (as defined by the user agent),

and the cookie's "secure-only-flag" is "true", then abort these steps and ignore the newly created cookie entirely.

2. Before step 11, execute the following step:

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

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

Note: This comparison intentionally ignores the "path" component. The intent is to allow the "secure" flag to supercede the "path" restrictions to protect sites against cookie fixing attacks.

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

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

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

Note that the eviction algorithm specified here is triggered only after insertion of a cookie which causes the user agent to exceed some predetermined upper bound. Conforming user agents MUST

ensure that inserting a non-secure cookie does not cause a secure cookie to be removed.

4. Security Considerations

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

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

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

5. References

5.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<http://www.rfc-editor.org/info/rfc3986>>.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <<http://www.rfc-editor.org/info/rfc6265>>.

5.2. Informative References

[COOKIE-INTEGRITY]

Zheng, X., Jiang, J., Liang, J., Duan, H., Chen, S., Wan, T., and N. Weaver, "Cookies Lack Integrity: Real-World Implications", n.d., <<https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-zheng.pdf>>.

[COOKIE-PREFIXES]

West, M., "Cookie Prefixes", n.d., <<https://tools.ietf.org/html/draft-west-cookie-prefixes>>.

[HSTS-PRELOADING]

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

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

Appendix A. Acknowledgements

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

Author's Address

Mike West
Google, Inc

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