HTTPBis Working Group
Internet-Draft
Intended status: Informational
Expires: September 15, 2016

M. Bishop
Microsoft

March 14, 2016

Decomposing the Hypertext Transfer Protocol
draft-bishop-httpbis-decomposing-http-00

Abstract

The Hypertext Transfer Protocol in its various versions combines concepts of both an application and transport-layer protocol. As this group contemplates employing alternate transport protocols underneath HTTP, this document attempts to delineate the boundaries between these functions to define a shared vocabulary in discussing the revision and/or replacement of one or more of these components.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 15, 2016.

Table of Contents

1. Introduction

The Hypertext Transfer Protocol defines a very flexible tool set enabling client applications to make requests of a server for content or action. This general protocol was conceived for "the web," interconnected pages of Hypertext Markup Language (HTML) and associated resources used to render the HTML, but has since been used as a general-purpose application transport. Server APIs are commonly exposed as REST APIs, accessed over HTTP.

HTTP/1.0 [RFC1945] was a text-based protocol which did not specify its underlying transport, but describes the mapping this way:

> On the Internet, HTTP communication generally takes place over TCP/IP connections. The default port is TCP 80, but other ports can be used. This does not preclude HTTP from being implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used, and the mapping of the HTTP/1.0 request and response structures onto the transport data units of the protocol in question is outside the scope of this specification.

HTTP/1.1 [RFC7230] expands on the TCP binding, introducing connection management concepts into the HTTP layer.

HTTP/2 [RFC7540] replaced the simple text-based protocol with a binary framing. Conceptually, HTTP/2 achieved the same properties required of a TCP mapping using wildly different strategies from HTTP/1.1. HTTP/1.1 achieves properties such as parallelism and out-of-order delivery by the use of multiple TCP connections. HTTP/2 implements these services on top of TCP to enable the use of a single TCP connection. The working group's charter to maintain HTTP's broad applicability meant that there were few or no changes in how HTTP surfaces to applications.

Other efforts have mapped HTTP or a subset of it to various transport protocols besides TCP - HTTP can be implemented over SCTP [RFC4960] as in [I-D.natarajan-http-over-sctp], and useful profiles of HTTP have been mapped to UDP in various ways (HTTPU and HTTPUM in [goland-http-udp] and [UPnP], CoAP [RFC7252], QUIC [I-D.tsvwg-quic-protocol]).

With the publication of HTTP/2 over TCP, the working group is beginning to consider how a mapping to a non-TCP transport would function. This document aims to enable this conversation by describing the services required by the HTTP semantic layer. A mapping of HTTP to a transport other than TCP must define how these services are obtained, either from the new transport or by implementing them at the application layer.

2. The Semantic Layer

At the most fundamental level, the semantic layer of HTTP consists of a client's ability to request some action of a server and be informed of the outcome of that request. HTTP defines a number of possible actions (methods) the client might request of the server, but permits the list of actions to be extended.

A client's request consists of a desired action (HTTP method) and a resource on which that action is to be taken (path). The server

responds which a status code which informs the client of the result of the request - the outcome of the action or the reason the action was not performed. Actions may or may not be idempotent or safe, and the results may or may not be cached by intermediaries; this is defined as part of the HTTP method.

Each message (request or response) has associated metadata, called "headers," which provide additional information about the operation. In a request this might include client identification, credentials authorizing the client to request the action, or preferences about how the client would prefer the server handle the action. In a response, this might include information about the resulting data, modifications to the cacheability of the response, details about how the server performed the action, or details of the reason the server declined to perform the action.

The headers are key-value pairs, with rules defining how keys which occur multiple times should be handled. Due to artifacts of existing usage, these rules vary from key to key. For similar legacy reasons, there is no uniform structure of the values across all keys. Keys are case-insensitive ASCII strings, while values are sequences of octets typically interpreted as ASCII. Many headers are defined by the HTTP RFCs, but the space is not constrained and is frequently extended with little or no notice. "Trailing" headers are split, with the key declared in advance, but the value coming only after the body has been transferred.

Each message, whether request or response, also has an optional body. The presence and content of the body will vary based on the action requested and the headers provided.

3. Transport Services Required

The HTTP Semantic Layer depends on the availability of several services from its lower layer:

- Reliable delivery
- In-order delivery
- Partial delivery
- Separate request/response, metadata, and payload
- Flow control and throttling

In this section, each of these properties will be discussed at a high level with a focus on why HTTP requires these properties to be

present. The next section (Section 4) will discuss how various HTTP mappings have handled the absence of these required services in different transports.

3.1. Reliable delivery

HTTP does not provide the concept to higher layers that fragments of data were received while others were not. If a request is sent, it is assumed that either a response will arrive or the transport will report an error. HTTP itself is not concerned with any intermediate states.

There are many ways for a transport to provide reliable delivery of messages. This may take the form of loss recovery, where the loss of packets is detected and the corresponding information retransmitted. Alternately, a transport may proactively send extra information so that the data stream is tolerant to some loss - the full message can be reconstructed after receipt of a sufficient fraction of the transmission.

It is worth noting that some consumers of HTTP have relaxed requirements in this space - while HTTP itself has no notion of lossy delivery, some mappings do have weakened guarantees and are only appropriate for scenarios where those weakened guarantees are acceptable.

3.2. In-order delivery

The headers of each message must arrive before any body, since they dictate how the body will be processed. The body is typically exposed as a bytestream which can be read from sequentially, though there are some consumers who are able to use incomplete fragments of certain resource types.

Regardless of the ability to surface and use fragmentary pieces of an HTTP message, the HTTP layer requires the transport be able to ultimately provide a correct ordering and full reconstruction of each message.

3.3. Partial delivery

While only some users of HTTP (client or server) are able to deal with unordered fragments of an HTTP message, it is almost universally necessary to deal with HTTP messages in pieces. There are multiple reasons why that may be necessary:

- The message may be too large to maintain in memory at once (the download of a large file)

- The beginning of a request may be sufficient to generate a response (error due to lack of authorization)
- The message may be constructed incrementally, sending each segment as it becomes available

Regardless, HTTP needs the transport to begin sending the message before the end of the message is available.

3.4. Separate request/response, metadata, and payload

Any protocol defines how the semantics of the protocol are mapped onto the wire in a transport. Most transports are either bytestreams or message-based, meaning that higher-layer concepts must be laid out in a reasonable structure within the stream or message. Each HTTP request or response contains metadata about the message (headers) and an optional body.

These are separate constructs in HTTP, and mechanisms to carry them and keep them appropriately associated must be provided. Note that it's not actually expected that any _generic_ transport layer would or should have this property, but is nonetheless involved in transporting HTTP messages.

3.5. Flow control and throttling

Flow control is a necessary property of any transport. Because no network can handle an uncontrolled burst of data at infinite speeds, the transport must determine an appropriate sustained data rate for the intervening network. Even in the presence of a nearly-infinite network capacity, the remote server will also have limits on its ability to consume data.

In order to avoid overwhelming either the network or the server, HTTP requires a mechanism to limit sending data rates as well as to limit the rate of new requests going to a server. Although it is optimal for a server to know about all outstanding client requests (even if it chooses not to work on them immediately), the server may wish to protect itself by limiting the memory commitment to outstanding data or requests. The transport should facilitate such protection on the part of a server (or client, in certain scenarios).

3.6. Other desirable properties

There are several properties not properly required for the implementation of HTTP, but which users of HTTP have come to assume are present.

3.6.1. Parallelism

Because a client will often desire a single server to perform multiple actions at once, all HTTP mappings provide the ability to deliver requests in parallel and allow the server to respond to each request as the actions complete. Head-of-line blocking is a particular problem here that transports must attempt to avoid - client requests should ideally reach the server as quickly as possible, allowing the server to choose the correct order in which to handle the requests (with input from the client). Any situation in which a request remains unknown to the server until another request completes is suboptimal.

3.6.2. Security

Integrity and confidentiality are valuable services for communication over the Internet, and HTTP is no exception. While authentication, message integrity, and secrecy are not inherently _required_ for the implementation of HTTP, they are advantageous properties for any mapping to have, so that each party can be sure that what they received is what the other party sent.

Privacy, the control of what data is leaked to the peer and/or third parties, is also a desirable attribute. However, this extends well beyond the scope of any particular mapping and into the use of HTTP.

TLS [RFC5246] is commonly used in mappings to provide this service, and itself requires reliable, in-order delivery. When those services are not provided by the underlying transport, the mapping must either provide those services to TLS as well as HTTP (as in QUIC) or a variant of TLS which provides those services for itself must be substituted (DTLS [RFC6347], as used in CoAP).

3.6.3. Efficiency

While it would be technically possible to define HTTP over a highly inefficient transport or mapping (e.g. format messages in Baudot code, transporting them to the server using avian carriers as in [RFC1149]), there is little reason for applications to use such inefficient mappings when efficient transport mappings exist.

Efficiency can be characterized on many levels:

- Reducing the number of bytes required to transport a message, either through lower overhead or better compression
- Reducing the time from request generation to response receipt

- o Reducing the amount of computation or memory required to process or route a request

- o Reducing the power consumption required to generate or process a request

4. The Transport Adaptation Layer

No present transport over which HTTP has been mapped actually provides all of the services on which the HTTP Semantic Layer depends. In order to compensate for the services not provided by a given underlying transport, each mapping of HTTP onto a new transport must define an intermediate layer implementing the missing services in order to enable the mapping, as well as any additional features the mapping finds to be desirable.

In the following table, we can see multiple transports over which HTTP has been deployed and the services which the underlying transports do or do not offer.

	TCP	UDP	SCTP	QUIC
Reliable delivery	X		X	X
In-order delivery	X		X	X
Partial delivery	X	X	X	X
Separate metadata and payload				*
Flow control & throttling	X	X	X	X

Some mappings contain entirely new protocol machinery constructed specifically to serve as an adaptation layer and carried within the transport (HTTP/2 framing over TCP). Others rely on implementation-level meta-protocol behavior (simultaneous TCP connections handled in parallel) not visible to the transport. Because the existence of these adaptation layers has not been explicitly defined in the past, a clean separation has not always been maintained between the adaptation layer and either the transport or the semantic layer.

Some adaptation layers are so complex and fully-featured that the transport layer plus the adaptation layer can be conceptually treated as a new transport. For example, QUIC was originally designed as a transport adaptation layer for HTTP over UDP, but is now being refactored into a general-purpose transport layer for arbitrary

protocols. Such a refactoring will require separating the services QUIC provides that are general to all applications from the services which exist purely to enable a mapping of HTTP to QUIC. (In the table above, QUIC is referenced as a generic transport; the HTTP-over-QUIC mapping is discussed below.)

4.1. HTTP/1.x over TCP

Since HTTP/1.x is defined over TCP, many of the necessary services are provided by the transport, enabling a relatively simple mapping. However, there were a number of conventions introduced to fill lacks in the underlying transport.

4.1.1. Metadata and framing

HTTP/1.x projects a message as an octet sequence which typically resembles a block of ASCII text. Specific octets are used to delimit the boundaries between message components. Within the portion of the message dedicated to headers, the key-value pairs are expressed as text, with the ':' character and whitespace separating the key from the value.

Because this region appears to be text, many text conventions have accidentally crept into HTTP/1.x message parsers and even protocol conventions (line-folding, CRLF differences between operating systems, etc.). This is a source of bugs, such as line-folding characters which appear in header values even after being unframed.

4.1.2. Parallelism and request limiting

HTTP/1.0 used a very simple multi-request model - each request was made on a separate TCP connection, and all requests were handled independently. This had the drawback that TCP connection setup was required with each request and flow control almost never exited the slow-start phase, limiting performance.

To improve this, new headers were introduced to manage connection lifetime (e.g. "Connection: keep-alive"), blurring the distinction between message metadata and connection metadata. These headers were formalized in HTTP/1.1. This improvement means that connections are reused - when the end of a response has been received, a new request can be sent. However, this blurring made it difficult for some implementations to correctly identify the presence and length of bodies, making request-smuggling attacks possible as in [watchfire-request-smuggling].

Throttling of simultaneous requests was fully in the realm of implementations, which constrained themselves to opening only a

limited number of connections. HTTP/1.1 originally recommended two, but later implementations increased this to six by default, and more under certain conditions. Because these were fully independent flows, TCP was unable to consider them as a group for purposes of congestion control, leading to suboptimal behavior on the network.

Servers which desired additional parallelism could game such implementations by exposing resources under multiple hostnames, causing the client implementations to open six connections _to each hostname_ and gain an arbitrary amount of parallelism, to the detriment of functional congestion control.

4.1.3. Security

HTTP originally defined no additional integrity or confidentiality mechanisms for the TCP mapping, leaving the integrity and confidentiality levels to those provided by the network transport. These may be minimal (TCP checksums) or rich (IPsec) depending on the network environment.

For situations where the network does not provide integrity and confidentiality guarantees sufficient to the content, [RFC2818] defines the use of TLS as an additional component of the adaptation layer in HTTP/1.1.

4.1.4. Attempts to improve the TCP mapping

Pipelining, also introduced in HTTP/1.1, allowed the client to eliminate the round-trip that was incurred between the end of the server's response to one request and the server's receipt of the client's next request. However, pipelining increases the problem of head-of-line blocking since a request on a different connection might complete sooner. The client's inability to predict the length of requested actions limited the usefulness of pipelining.

SMUX [w3c-smux] allowed the use of a single TCP connection to carry multiple channels over which HTTP could be carried. This would permit the server to answer requests in any order. However, this was never broadly deployed.

4.2. HTTP/1.x over SCTP

Because SCTP permits the use of multiple simultaneous streams over a single connection, HTTP/1.1 could be mapped with relative ease. Instead of using separate TCP connections, SCTP flows could be used to provide a multiplexing layer. Each flow was reused for new requests after the completion of a response, just as HTTP/1.1 used

TCP connections. This allowed for better flow control performance, since the transport could consider all flows together.

SCTP has seen limited deployment on the Internet, though recent experience has shown SCTP over UDP [RFC6951] to be a more viable combination.

4.3. HTTP/2 over TCP

HTTP/2, also a TCP mapping, attempted to improve the mapping of HTTP to TCP without introducing changes at the semantic level.

> HTTP/2 addresses these issues by defining an optimized mapping of HTTP's semantics to an underlying connection. Specifically, it allows interleaving of request and response messages on the same connection and uses an efficient coding for HTTP header fields. It also allows prioritization of requests, letting more important requests complete more quickly, further improving performance.
>
> The resulting protocol is more friendly to the network because fewer TCP connections can be used in comparison to HTTP/1.x. This means less competition with other flows and longer-lived connections, which in turn lead to better utilization of available network capacity.
>
> Finally, HTTP/2 also enables more efficient processing of messages through use of binary message framing.

4.3.1. Framing and Parallelism

HTTP/2 introduced a framing layer that incorporated the concept of streams. Because a very large number of idle streams automatically exist at the beginning of each connection, each stream can be used for a single request and response. One stream is dedicated to the transport of control messages, enabling a cleaner separation between metadata about the connection from metadata about the separate messages within the connection.

HTTP/2 projects the requested action into the set of headers, then uses separate HEADERS and DATA frames to delimit the boundary between metadata and message body on each stream. These frames are used to provide message-like behaviors and parallelism over a single TCP bytestream.

Because the text-based transfer of repetitive headers represented a major inefficiency in HTTP/1.1, HTTP/2 also introduced HPACK [RFC7541], a custom compression scheme which operates on key-value pairs rather than text blocks. HTTP/2 frame types which transport

headers always carry HPACK header block fragments rather than an uncompressed key-value dictionary.

4.3.2. Congestion and flow control

Because HTTP/2's adaptation layer introduces a concurrency construct above the transport, the adaptation layer must also introduce a means of flow control to keep the concurrent transactions from introducing head-of-line blocking above TCP. This led HTTP/2 to create a flow-control scheme within the adaptation layer in addition to TCP's flow control algorithms.

In HTTP/1.1, this was not needed - the application simply reads from TCP as space is available, and allow's TCP's own flow control to govern. In HTTP/2, this would cause severe head-of-line blocking due to the increased parallelism, and so the control must be exerted at a higher level.

Another drawback to the application-layer multiplexing approach is the fact that TCP's congestion-avoidance mechanisms cannot identify the flows separately, magnifying the impact of packet losses. This manifests both by reducing the congestion window for the entire connection (versus one-sixth of the "connection" in HTTP/1.1) on packet loss, and delayed delivery of packets on unaffected streams due to head-of-line blocking behind lost packets.

4.3.3. Security

HTTP/2 directly defines how TLS may be used to provide security services as part of its adaptation layer.

4.4. HTTPU(M) and CoAP

UDP mappings of HTTP must define mechanisms to restore the original order of message fragments. HTTPU(M) and the base form of CoAP both do this by restricting messages to the size of a single datagram, while [I-D.ietf-core-block] extends CoAP to define an in-order delivery mechanism in the adaptation layer.

Adaptation layers of HTTP mappings over UDP have also needed to introduce mechanisms for reliable delivery. CoAP dedicates a portion of its message framing to indicating whether a given message requires reliability or not. If reliable delivery is required, the recipient acknowledges receipt and the sender continues to repeat the message until the acknowledgment is received. For non-idempotent requests, this means keeping additional state about which requests have already been processed.

Some applications above HTTP are able to provide their own loss-recovery messages, and therefore do not actually require the guarantees that HTTP provides. HTTP over UDP Multicast is targeted at such applications, and therefore does not provide reliable delivery to applications above it.

4.5. QUIC over UDP, or HTTP/2 over QUIC, or...?

QUIC is an overloaded term. QUIC is a rich HTTP mapping to UDP [I-D.tsvwg-quic-protocol] which implements many TCP- and SCTP-like behaviors in its adaptation layer. It describes itself this way:

> QUIC (Quick UDP Internet Connection) is a new multiplexed and secure transport atop UDP, designed from the ground up and optimized for HTTP/2 semantics. While built with HTTP/2 as the primary application protocol, QUIC builds on decades of transport and security experience, and implements mechanisms that make it attractive as a modern general-purpose transport. QUIC provides multiplexing and flow control equivalent to HTTP/2, security equivalent to TLS, and connection semantics, reliability, and congestion control equivalent to TCP.

Consequently, QUIC is _also_ a "general-purpose transport" over which an HTTP mapping can be defined and implemented.

This division makes it unclear which parts belong to the transport versus an HTTP mapping on top of this new transport. For example, [I-D.tsvwg-quic-protocol] does define how to separately transport the headers and body of an HTTP message. However, this capability is likely not relevant in a general-purpose transport and might better be removed from QUIC-the-transport and incorporated into HTTP-over-QUIC.

5. Moving Forward

The networks over which we run TCP/IP today look nothing like the networks for which TCP/IP was originally designed. It is the clean separation between TCP, IP, and the lower-layer protocols which has enabled the continued usefulness of the higher-layer protocols as the substrate has changed. Likewise, the actions and content carried over HTTP look very different, reflecting well on the abstraction achieved by the HTTP layer.

It is the layer between HTTP and the transport where abstraction has not always been successfully achieved. New capabilites in transports have required new expressions at the HTTP layer to take advantage of them, and mappings have defined concepts which are tightly bound to

the underlying transport without clearly separating them from the semantics of HTTP.

The goal is not merely architectural purity, but modularity. HTTP has enjoyed a long life as a higher-layer protocol and is useful to many varied applications. As transports continue to evolve, we will almost certainly find ourselves in the position of defining a mapping of HTTP onto a new transport once again. With a clear understanding of the HTTP semantic layer and the services it requires, we can better scope the requirements of a new adaptation layer while reusing the components of previous adaptation layers that provide the necessary service well in existing implementations.

6. Informative References

[goland-http-udp]
Goland, Y., "Multicast and Unicast UDP HTTP Messages", November 1999, <http://tools.ietf.org/html/draft-goland-http-udp-01>.

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.

[I-D.natarajan-http-over-sctp]
Natarajan, P., Amer, P., Leighton, J., and F. Baker, "Using SCTP as a Transport Layer Protocol for HTTP", draft-natarajan-http-over-sctp-02 (work in progress), July 2009.

[I-D.tsvwg-quic-protocol]
Hamilton, R., Iyengar, J., Swett, I., and A. Wilk, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2", draft-tsvwg-quic-protocol-02 (work in progress), January 2016.

[RFC1149] Waitzman, D., "Standard for the transmission of IP datagrams on avian carriers", RFC 1149, DOI 10.17487/RFC1149, April 1990, <http://www.rfc-editor.org/info/rfc1149>.

[RFC1945] Berners-Lee, T., Fielding, R., and H. Frystyk, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, DOI 10.17487/RFC1945, May 1996, <http://www.rfc-editor.org/info/rfc1945>.

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

[RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <http://www.rfc-editor.org/info/rfc4960>.

[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>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6951] Tuexen, M. and R. Stewart, "UDP Encapsulation of Stream Control Transmission Protocol (SCTP) Packets for End-Host to End-Host Communication", RFC 6951, DOI 10.17487/RFC6951, May 2013, <http://www.rfc-editor.org/info/rfc6951>.

[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>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

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

[RFC7541] Peon, R. and H. Ruellan, "HPACK: Header Compression for HTTP/2", RFC 7541, DOI 10.17487/RFC7541, May 2015, <http://www.rfc-editor.org/info/rfc7541>.

[UPnP] "UPnP Device Architecture 2.0", 2015, <http://upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v2.0.pdf>.

[w3c-smux]
Nielsen, H., "SMUX Protocol Specification", July 1998, <http://www.w3.org/TR/WD-mux>.

[watchfire-request-smuggling]
Orrin, S., "HTTP Request Smuggling", 2005, <http://www.cgisecurity.com/lib/HTTP-Request-Smuggling.pdf>.

Author's Address

Mike Bishop
Microsoft

Email: michael.bishop@microsoft.com

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

M. Bishop

N. Sullivan
Cloudflare
M. Thomson
Mozilla

October 30, 2017

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

Abstract

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

Many HTTP servers host content from several origins. HTTP/2 [RFC7540] permits clients to reuse an existing HTTP connection to a server provided that the secondary origin is also in the certificate provided during the TLS [I-D.ietf-tls-tls13] handshake.

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

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

Status of This Memo

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

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

Copyright Notice

Table of Contents

1. Introduction

HTTP clients need to know that the content they receive on a connection comes from the origin that they intended to retrieve in from. The traditional form of server authentication in HTTP has been in the form of X.509 certificates provided during the TLS RFC5246 [I-D.ietf-tls-tls13] handshake.

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

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

1.1. Server Certificate Authentication

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

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

[I-D.ietf-httpbis-origin-frame] relaxes the requirement to perform the DNS lookup if already connected to a server with an appropriate certificate which claims support for a particular origin.

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

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

1.2. Client Certificate Authentication

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

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

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

1.2.1. HTTP/1.1 using TLS 1.2 and previous

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

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

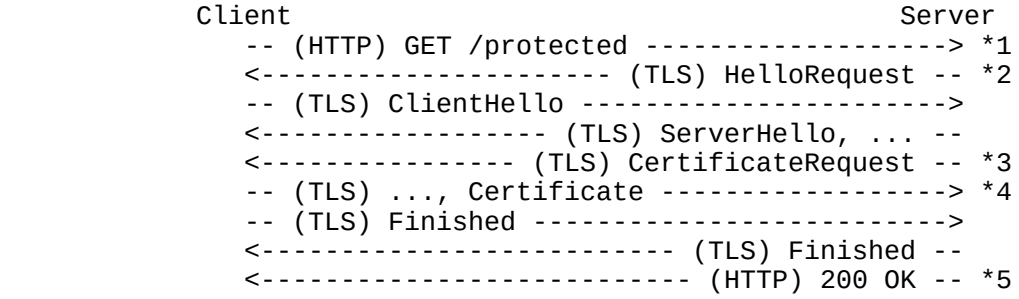


Figure 1: HTTP/1.1 Reactive Certificate Authentication with TLS 1.2

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

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

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

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

1.2.2. HTTP/1.1 using TLS 1.3

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

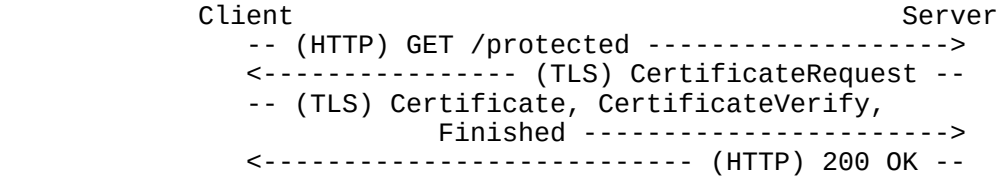


Figure 2: HTTP/1.1 Reactive Certificate Authentication with TLS 1.3

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

1.2.3. HTTP/2

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

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

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

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

renegotiation. This is problematic for a multiplexed protocol like HTTP/2.

1.3. HTTP-Layer Certificate Authentication

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

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

TLS Exported Authenticators are structured messages that can be exported by either party of a TLS connection and validated by the other party. An authenticator message can be constructed by either the client or the server given an established TLS connection, a certificate, and a corresponding private key. Exported Authenticators use the message structures from section 4.4 of [I-D.ietf-tls-tls13], but different parameters.

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

Successfully verified Authenticators result in certificate chains, with verified possession of the corresponding private key, which can be supplied into a collection of available certificates. Likewise, descriptions of desired certificates can be supplied into these collections. These pre-supplied elements are then available for automatic use (in some situations) or for reference by individual streams.

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

1.4. Terminology

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

2. Discovering Additional Certificates at the HTTP/2 Layer

A certificate chain with proof of possession of the private key corresponding to the end-entity certificate is sent as a single "CERTIFICATE" frame (see Section 3.4) on stream zero. Once the holder of a certificate has sent the chain and proof, this certificate chain is cached by the recipient and available for future use. If the certificate is marked as "AUTOMATIC_USE", the certificate may be used by the recipient to authorize any current or future request. Otherwise, the recipient requests the required certificate on each stream, but the previously-supplied certificates are available for reference without having to resend them.

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

Data sent by each peer is correlated by the ID given in each frame. This ID is unrelated to values used by the other peer, even if each uses the same ID in certain cases.

2.1. Indicating support for HTTP-layer certificate authentication

Clients and servers that will accept requests for HTTP-layer certificate authentication indicate this using the HTTP/2 "SETTINGS_HTTP_CERT_AUTH" (0xSETTING-TBD) setting.

The initial value for the "SETTINGS_HTTP_CERT_AUTH" setting is 0, indicating that the peer does not support HTTP-layer certificate authentication. If a peer does support HTTP-layer certificate authentication, the value is 1.

2.2. Making certificates or requests available

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

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

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

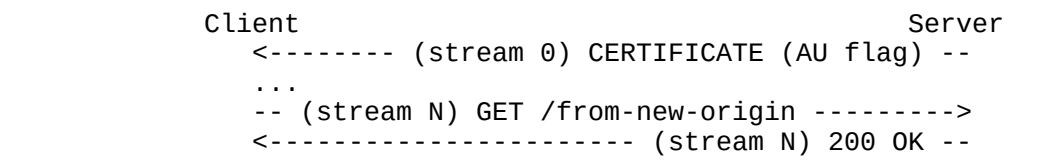


Figure 3: Proactive Server Certificate

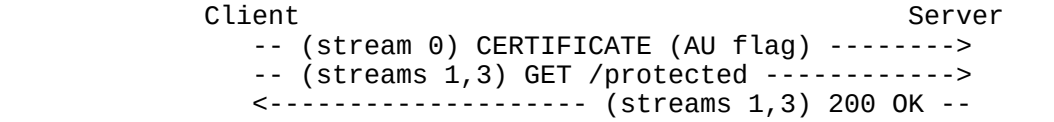


Figure 4: Proactive Client Certificate

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

2.3. Requiring certificate authentication

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

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

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

If the server does not have the desired certificate, it MUST respond with an empty "USE_CERTIFICATE" frame. In this case, or if the server has not advertised support for HTTP-layer certificates, the client MUST NOT send any requests for resources in that origin on the current connection.

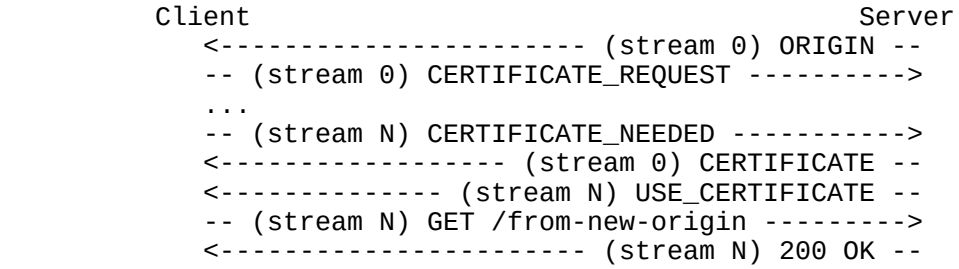


Figure 5: Client-Requested Certificate

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

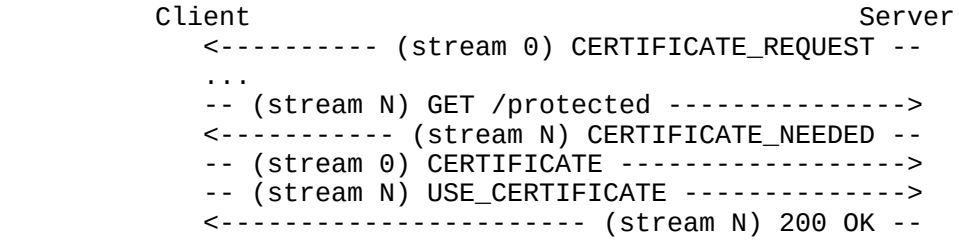


Figure 6: Reactive Certificate Authentication

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

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

3. Certificates Frames for HTTP/2

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

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

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

3.1. The CERTIFICATE_NEEDED frame

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

The "CERTIFICATE_NEEDED" frame contains 2 octets, which is the authentication request identifier, "Request-ID". A peer that receives a "CERTIFICATE_NEEDED" of any other length MUST treat this as a stream error of type "PROTOCOL_ERROR". Frames with identical request identifiers refer to the same "CERTIFICATE_REQUEST".

A server MAY send multiple "CERTIFICATE_NEEDED" frames on the same stream. If a server requires that a client provide multiple certificates before authorizing a single request, each required certificate MUST be indicated with a separate "CERTIFICATE_NEEDED" frame, each of which MUST have a different request identifier (referencing different "CERTIFICATE_REQUEST" frames describing each

required certificate). To reduce the risk of client confusion, servers SHOULD NOT have multiple outstanding "CERTIFICATE_NEEDED" frames on the same stream at any given time.

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

The "CERTIFICATE_NEEDED" frame MUST NOT be sent to a peer which has not advertised support for HTTP-layer certificate authentication.

The "CERTIFICATE_NEEDED" frame MUST NOT be sent on stream zero, and MUST NOT be sent on a stream in the "half-closed (local)" state [RFC7540]. A client that receives a "CERTIFICATE_NEEDED" frame on a stream which is not in a valid state SHOULD treat this as a stream error of type "PROTOCOL_ERROR".

3.2. The USE_CERTIFICATE Frame

The "USE_CERTIFICATE" frame (0xFRAME-TBD4) is sent in response to a "CERTIFICATE_NEEDED" frame to indicate which certificate is being used to satisfy the requirement.

A "USE_CERTIFICATE" frame with no payload refers to the certificate provided at the TLS layer, if any. If no certificate was provided at the TLS layer, the stream should be processed with no authentication, likely returning an authentication-related error at the HTTP level (e.g. 403) for servers or routing the request to a new connection for clients.

Otherwise, the "USE_CERTIFICATE" frame contains the two-octet "Cert-ID" of the certificate the sender wishes to use. This MUST be the ID of a certificate for which proof of possession has been presented in a "CERTIFICATE" frame. Recipients of a "USE_CERTIFICATE" frame of any other length MUST treat this as a stream error of type "PROTOCOL_ERROR". Frames with identical certificate identifiers refer to the same certificate chain.

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

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

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

If these requirements are not satisfied, the recipient MAY at its discretion either return an error at the HTTP semantic layer, or respond with a stream error [RFC7540] on any stream where the certificate is used. Section 4 defines certificate-related error codes which might be applicable.

3.3. The CERTIFICATE_REQUEST Frame

TLS 1.3 defines the "CertificateRequest" message, which prompts the client to provide a certificate which conforms to certain properties specified by the server. This draft defines the "CERTIFICATE_REQUEST" frame (0xFRAME-TBD2), which uses the same set of extensions to specify a desired certificate, but can be sent over any TLS version and can be sent by either peer.

The "CERTIFICATE_REQUEST" frame SHOULD NOT be sent to a peer which has not advertised support for HTTP-layer certificate authentication.

The "CERTIFICATE_REQUEST" frame MUST be sent on stream zero. A "CERTIFICATE_REQUEST" frame received on any other stream MUST be rejected with a stream error of type "PROTOCOL_ERROR".

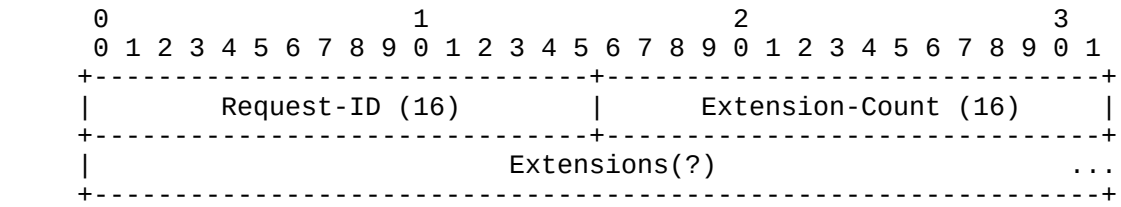


Figure 7: CERTIFICATE_REQUEST frame payload

The frame contains the following fields:

Request-ID: "Request-ID" is a 16-bit opaque identifier used to correlate subsequent certificate-related frames with this request. The identifier MUST be unique in the session for the sender.

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

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

Servers MUST be able to recognize the "server_name" extension ([RFC6066]) at a minimum. Clients MUST always specify the desired origin using this extension, though other extensions MAY also be included.

3.4. The CERTIFICATE Frame

The "CERTIFICATE" frame (id=0xFRAME-TBD3) provides a exported authenticator message from the TLS layer that provides a chain of certificates, associated extensions and proves possession of the private key corresponding to the end-entity certificate.

The "CERTIFICATE" frame defines two flags:

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

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

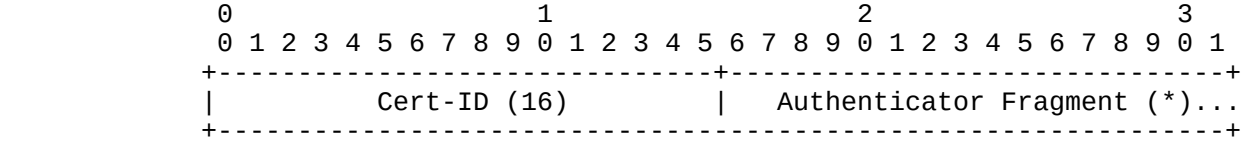


Figure 8: CERTIFICATE frame payload

The "Exported Authenticator Fragment" field contains a portion of the opaque data returned from the TLS connection exported authenticator "authenticate" API. See Section 3.4.1 for more details on the input to this API.

This opaque data is transported in zero or more "CERTIFICATE" frames with the "TO_BE_CONTINUED" flag set, followed by one "CERTIFICATE" frame with the "TO_BE_CONTINUED" flag unset. Each of these frames contains the same "Cert-ID" field, permitting them to be associated with each other. Receipt of any "CERTIFICATE" frame with the same "Cert-ID" following the receipt of a "CERTIFICATE" frame with "TO_BE_CONTINUED" unset MUST be treated as a connection error of type "PROTOCOL_ERROR".

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

Upon receiving a complete series of "CERTIFICATE" frames, the receiver may validate the Exported Authenticator value by using the exported authenticator API. This returns either an error indicating that the message was invalid, or the certificate chain and extensions used to create the message.

The "CERTIFICATE" frame MUST be sent on stream zero. A "CERTIFICATE" frame received on any other stream MUST be rejected with a stream error of type "PROTOCOL_ERROR".

3.4.1. Exported Authenticator Characteristics

The Exported Authenticator API defined in [I-D.ietf-tls-exported-authenticator] takes as input a certificate, supporting information about the certificate (OCSP, SCT, etc.), and an optional "certificate_request_context". When generating exported authenticators for use with this extension, the "certificate_request_context" MUST be the two-octet Cert-ID.

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

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

Once the authenticator is accepted, the endpoint can perform any other checks for the acceptability of the certificate itself.

4. Indicating failures during HTTP-Layer Certificate Authentication

Because this draft permits certificates to be exchanged at the HTTP framing layer instead of the TLS layer, several certificate-related errors which are defined at the TLS layer might now occur at the HTTP

framing layer. In this section, those errors are restated and added to the HTTP/2 error code registry.

BAD_CERTIFICATE (0xERROR-TBD1): A certificate was corrupt, contained signatures that did not verify correctly, etc.

UNSUPPORTED_CERTIFICATE (0xERROR-TBD2): A certificate was of an unsupported type or did not contain required extensions

CERTIFICATE_REVOKED (0xERROR-TBD3): A certificate was revoked by its signer

CERTIFICATE_EXPIRED (0xERROR-TBD4): A certificate has expired or is not currently valid

CERTIFICATE_GENERAL (0xERROR-TBD5): Any other certificate-related error

As described in [RFC7540], implementations MAY choose to treat a stream error as a connection error at any time. Of particular note, a stream error cannot occur on stream 0, which means that implementations cannot send non-session errors in response to "CERTIFICATE_REQUEST", and "CERTIFICATE" frames. Implementations which do not wish to terminate the connection MAY either send relevant errors on any stream which references the failing certificate in question or process the requests as unauthenticated and provide error information at the HTTP semantic layer.

5. Security Considerations

This mechanism defines an alternate way to obtain server and client certificates other than in the initial TLS handshake. While the signature of exported authenticator values is expected to be equally secure, it is important to recognize that a vulnerability in this code path is at least equal to a vulnerability in the TLS handshake.

5.1. Impersonation

This mechanism could increase the impact of a key compromise. Rather than needing to subvert DNS or IP routing in order to use a compromised certificate, a malicious server now only needs a client to connect to _some_ HTTPS site under its control in order to present the compromised certificate. As recommended in [I-D.ietf-httpbis-origin-frame], clients opting not to consult DNS ought to employ some alternative means to increase confidence that the certificate is legitimate.

As noted in the Security Considerations of [I-D.ietf-tls-exported-authenticator], it difficult to formally prove that an endpoint is jointly authoritative over multiple certificates, rather than individually authoritative on each certificate. As a result, clients MUST NOT assume that because one origin was previously colocated with another, those origins will be reachable via the same endpoints in the future. Clients MUST NOT consider previous secondary certificates to be validated after TLS session resumption. However, clients MAY proactively query for previously-presented secondary certificates.

5.2. Fingerprinting

This draft defines a mechanism which could be used to probe servers for origins they support, but opens no new attack versus making repeat TLS connections with different SNI values. Servers SHOULD impose similar denial-of-service mitigations (e.g. request rate limits) to "CERTIFICATE_REQUEST" frames as to new TLS connections.

While the extensions in the "CERTIFICATE_REQUEST" frame permit the sender to enumerate the acceptable Certificate Authorities for the requested certificate, it might not be prudent (either for security or data consumption) to include the full list of trusted Certificate Authorities in every request. Senders, particularly clients, SHOULD send only the extensions that narrowly specify which certificates would be acceptable.

5.3. Denial of Service

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

Validating a multitude of signatures can be computationally expensive, while generating an invalid signature is computationally cheap. Implementations will require checks for attacks from this direction. Invalid exported authenticators SHOULD be treated as a session error, to avoid further attacks from the peer, though an implementation MAY instead disable HTTP-layer certificates for the current connection instead.

5.4. Confusion About State

Implementations need to be aware of the potential for confusion about the state of a connection. The presence or absence of a validated certificate can change during the processing of a request, potentially multiple times, as "USE_CERTIFICATE" frames are received. A server that uses certificate authentication needs to be prepared to

reevaluate the authorization state of a request as the set of certificates changes.

Client implementations need to carefully consider the impact of setting the "AUTOMATIC_USE" flag. This flag is a performance optimization, permitting the client to avoid a round-trip on each request where the server checks for certificate authentication. However, once this flag has been sent, the client has zero knowledge about whether the server will use the referenced cert for any future request, or even for an existing request which has not yet completed. Clients MUST NOT set this flag on any certificate which is not appropriate for currently-in-flight requests, and MUST NOT make any future requests on the same connection which they are not willing to have associated with the provided certificate.

6. IANA Considerations

This draft adds entries in three registries.

The HTTP/2 "SETTINGS_HTTP_CERT_AUTH" setting is registered in Section 6.1. Four frame types are registered in Section 6.2. Six error codes are registered in Section 6.3.

6.1. HTTP/2 SETTINGS_HTTP_CERT_AUTH Setting

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

Name: SETTINGS_HTTP_CERT_AUTH

Code: 0xSETTING-TBD

Initial Value: 0

Specification: This document.

6.2. New HTTP/2 Frames

Four new frame types are registered in the "HTTP/2 Frame Types" registry established in [RFC7540]. The entries in the following table are registered by this document.

Frame Type	Code	Specification
CERTIFICATE_NEEDED	0xFRAME-TBD1	Section 3.1
CERTIFICATE_REQUEST	0xFRAME-TBD2	Section 3.3
CERTIFICATE	0xFRAME-TBD3	Section 3.4
USE_CERTIFICATE	0xFRAME-TBD4	Section 3.2

6.3. New HTTP/2 Error Codes

Five new error codes are registered in the "HTTP/2 Error Code" registry established in [RFC7540]. The entries in the following table are registered by this document.

Name	Code	Specification
BAD_CERTIFICATE	0xERROR-TBD1	Section 4
UNSUPPORTED_CERTIFICATE	0xERROR-TBD2	Section 4
CERTIFICATE_REVOKED	0xERROR-TBD3	Section 4
CERTIFICATE_EXPIRED	0xERROR-TBD4	Section 4
CERTIFICATE_GENERAL	0xERROR-TBD5	Section 4

7. Acknowledgements

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

8. References

8.1. Normative References

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

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

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

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

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

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

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

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

8.2. Informative References

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

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

Authors' Addresses

Mike Bishop

Email: mbishop@evequefou.be

Nick Sullivan
Cloudflare

Email: nick@cloudflare.com

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

HTTP Working Group
Internet-Draft
Intended status: 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

Table of Contents

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:

- 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.

- 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].

- 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:

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

- 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.

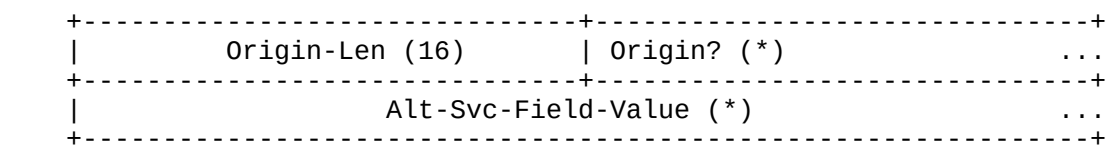
The ALTSVC frame is a non-critical extension to HTTP/2. Endpoints

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).



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:

- 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: Experimental
Expires: September 12, 2019

I. Grigorik
Google

March 11, 2019

HTTP Client Hints
draft-ietf-httpbis-client-hints-07

Abstract

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

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

It also defines an initial set of Client Hints.

Note to Readers

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

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/client-hints.

Status of This Memo

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

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

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

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

This Internet-Draft will expire on September 12, 2019.

Table of Contents

1. Introduction

There are thousands of different devices accessing the web, each with different device capabilities and preference information. These device capabilities include hardware and software characteristics, as well as dynamic user and client preferences.

One way to infer some of these capabilities is through User-Agent (Section 5.5.3 of [RFC7231]) header field detection against an established database of client signatures. However, this technique requires acquiring such a database, integrating it into the serving path, and keeping it up to date. However, even once this infrastructure is deployed, user agent sniffing has numerous limitations:

- User agent detection cannot reliably identify all static variables
- User agent detection cannot infer any dynamic client preferences
- User agent detection requires an external device database
- User agent detection is not cache friendly

A popular alternative strategy is to use HTTP cookies ([RFC6265]) to communicate some information about the user agent. However, this approach is also not cache friendly, bound by same origin policy, and often imposes additional client-side latency by requiring JavaScript execution to create and manage HTTP cookies.

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

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

This document defines a new response header, Accept-CH, that allows an origin server to explicitly ask that clients send these headers in requests, for a period of time bounded by the Accept-CH-Lifetime response header. It also defines guidelines for content negotiation mechanisms that use it, colloquially referred to as Client Hints.

Client Hints mitigate the performance concerns by assuring that clients will only send the request headers when they're actually

going to be used, and the privacy concerns of passive fingerprinting by requiring explicit opt-in and disclosure of required headers by the server through the use of the Accept-CH response header.

This document defines the Client Hints infrastructure, a framework that enables servers to opt-in to specific proactive content negotiation features, which will enable them to adapt their content accordingly. However, it does not define any specific features that will use that infrastructure. Those features will be defined in their respective specifications.

This document does not supersede or replace the User-Agent header field. Existing device detection mechanisms can continue to use both mechanisms if necessary. By advertising user agent capabilities within a request header field, Client Hints allow for cache friendly and proactive content negotiation.

1.1. Notational Conventions

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

This document uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with the list rule extension defined in [RFC7230], Appendix B. It includes by reference the DIGIT rule from [RFC5234] and the OWS and field-name rules from [RFC7230].

2. Client Hint Request Header Fields

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

2.1. Sending Client Hints

Clients control which Client Hints are sent in requests, based on their default settings, user configuration, and server preferences. The client and server can use an opt-in mechanism outlined below to negotiate which fields should be sent to allow for efficient content adaption, and optionally use additional mechanisms to negotiate delegation policies that control access of third parties to same fields.

Implementers should be aware of the passive fingerprinting implications when implementing support for Client Hints, and follow the considerations outlined in "Security Considerations" section of this document.

2.2. Server Processing of Client Hints

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

Further, depending on the hint used, the server can generate additional response header fields to convey related values to aid client processing.

2.2.1. Advertising Support via Accept-CH Header Field

Servers can advertise support for Client Hints using the Accept-CH header field or an equivalent HTML meta element with http-equiv attribute ([HTML5]).

```
Accept-CH = #field-name
```

For example:

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

When a client receives an HTTP response advertising support for Client Hints, it should process it as origin ([RFC6454]) opt-in to receive Client Hint header fields advertised in the field-value. The opt-in MUST be delivered over a secure transport.

For example, based on Accept-CH example above, a user agent could append the Sec-CH-Example and Sec-CH-Example-2 header fields to all same-origin resource requests initiated by the page constructed from the response.

2.2.2. The Accept-CH-Lifetime Header Field

Servers can ask the client to remember the set of Client Hints that the server supports for a specified period of time, to enable delivery of Client Hints on subsequent requests to the server’s origin ([RFC6454]).

```
Accept-CH-Lifetime = #delta-seconds
```

When a client receives an HTTP response that contains Accept-CH-Lifetime header field, the field-value indicates that the Accept-CH preference SHOULD be persisted and bound to the origin, and be considered stale after response's age ([RFC7234], section 4.2) is greater than the specified number of seconds. The preference MUST be delivered over a secure transport, and MUST NOT be persisted for an origin that isn't HTTPS.

```
Accept-CH: Sec-CH-Example, Sec-CH-Example-2
Accept-CH: Sec-CH-Example-3
Accept-CH-Lifetime: 86400
```

For example, based on the Accept-CH and Accept-CH-Lifetime example above, which is received in response to a user agent navigating to "https://example.com", and delivered over a secure transport: a user agent SHOULD persist an Accept-CH preference bound to "https://example.com" for up to 86400 seconds (1 day), and use it for user agent navigations to "https://example.com" and any same-origin resource requests initiated by the page constructed from the navigation's response. This preference SHOULD NOT extend to resource requests initiated to "https://example.com" from other origins.

If Accept-CH-Lifetime occurs in a message more than once, the last value overrides all previous occurrences.

2.2.3. Interaction with Caches

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

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

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

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

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

3. Security Considerations

The request header fields defined in this document, and those that extend it, expose information about the user's environment to enable proactive content negotiation. Such information may reveal new information about the user and implementers ought to consider the following considerations, recommendations, and best practices.

Transmitted Client Hints header fields SHOULD NOT provide new information that is otherwise not available to the application via other means, such as using HTML, CSS, or JavaScript. Further, sending highly granular data, such as image and viewport width may help identify users across multiple requests. Reducing the set of field values that can be expressed, or restricting them to an enumerated range where the advertised value is close but is not an exact representation of the current value, can improve privacy and reduce risk of linkability by ensuring that the same value is sent by multiple users. However, such precautions can still be insufficient for some types of data, especially data that can change over time.

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

- Implementers SHOULD restrict delivery of some or all Client Hints header fields to the opt-in origin only, unless the opt-in origin has explicitly delegated permission to another origin to request Client Hints header fields.
- Implementers MAY provide user choice mechanisms so that users may balance privacy concerns with bandwidth limitations. However, implementers should also be aware that explaining the privacy implications of passive fingerprinting to users may be challenging.
- Implementations specific to certain use cases or threat models MAY avoid transmitting some or all of Client Hints header fields. For example, avoid transmission of header fields that can carry higher risks of linkability.

Implementers SHOULD support Client Hints opt-in mechanisms and MUST clear persisted opt-in preferences when any one of site data, browsing history, browsing cache, or similar, are cleared.

4. IANA Considerations

This document defines the "Accept-CH" and "Accept-CH-Lifetime" HTTP response fields, and registers them in the Permanent Message Header Fields registry.

4.1. Accept-CH

- Header field name: Accept-CH
- Applicable protocol: HTTP
- Status: standard
- Author/Change controller: IETF
- Specification document(s): Section 2.2.1 of this document
- Related information: for Client Hints

4.2. Accept-CH-Lifetime

- Header field name: Accept-CH-Lifetime
- Applicable protocol: HTTP
- Status: standard
- Author/Change controller: IETF
- Specification document(s): Section 2.2.2 of this document
- Related information: for Client Hints

5. References

5.1. Normative References

[HTML5] Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, T., and S. Pfeiffer, "HTML5", World Wide Web Consortium Recommendation REC-html5-20141028, October 2014, <http://www.w3.org/TR/2014/REC-html5-20141028>.

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

[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>.

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

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

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

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

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

5.2. Informative References

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

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

Appendix A. Interaction with Key Response Header Field

Client Hints may be combined with Key response header field ([KEY]) to enable fine-grained control of the cache key for improved cache efficiency. For example, the server can return the following set of instructions:

```
Key: Sec-CH-Example;partition=1.5:2.5:4.0
```

Above example indicates that the cache key needs to include the value of the Sec-CH-Example header field with three segments: less than 1.5, 1.5 to less than 2.5, and 4.0 or greater.

```
Key: Width;Sec-CH-Example=320
```

Above example indicates that the cache key needs to include the value of the Sec-CH-Example header field and be partitioned into groups of 320: 0-320, 320-640, and so on.

Appendix B. Changes

B.1. Since -00

- Issue 168 (make Save-Data extensible) updated ABNF.
- Issue 163 (CH review feedback) editorial feedback from httpwg list.
- Issue 153 (NetInfo API citation) added normative reference.

B.2. Since -01

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

B.3. Since -02

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

B.4. Since -03

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

B.5. Since -04

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

B.6. Since -05

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

B.7. Since -06

- Issue 524: Save-Data is now defined by NetInfo spec, dropping

B.8. Since -07

- Removed specific features to be defined in other specifications

Acknowledgements

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

Author's Address

Ilya Grigorik
Google

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

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

M. West
Google, Inc

September 5, 2016

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

Abstract

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

Note to Readers

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on March 9, 2017.

Table of Contents

1. Introduction

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

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

2. Terminology and notation

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

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

3. Recommendations

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

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

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

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

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

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

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

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

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

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

4. Security Considerations

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

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

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

5. References

5.1. Normative References

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

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

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

5.2. Informative References

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

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

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

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

Appendix A. Acknowledgements

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

Appendix B. Changes

B.1. Since -00

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

Author's Address

Mike West
Google, Inc

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

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

M. West
Google, Inc

February 23, 2016

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

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on August 26, 2016.

Copyright Notice

Table of Contents

1. Introduction

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

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

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

2. Terminology and notation

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

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

3. Prefixes

3.1. The "__Secure-" prefix

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

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

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

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

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

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

3.2. The "__Host-" prefix

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

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

The following cookies would always be rejected:

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

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

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

4. User Agent Requirements

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

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

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

5. Aesthetic Considerations

5.1. Not pretty.

Prefixes are ugly. :(

5.2. Why "__"?

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

6. Security Considerations

6.1. Secure Origins Only

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

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

6.2. Limitations

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

7. References

7.1. Normative References

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

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

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

7.2. Informative References

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

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

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

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

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

Appendix A. Acknowledgements

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

Author’s Address

Mike West
Google, Inc

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

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

M. Thomson
Mozilla

April 18, 2017

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

Abstract

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

Note to Readers

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on October 20, 2017.

Table of Contents

1. Introduction

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

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

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

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

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

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

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

1.1. Notational Conventions

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

2. The "aes128gcm" HTTP Content Coding

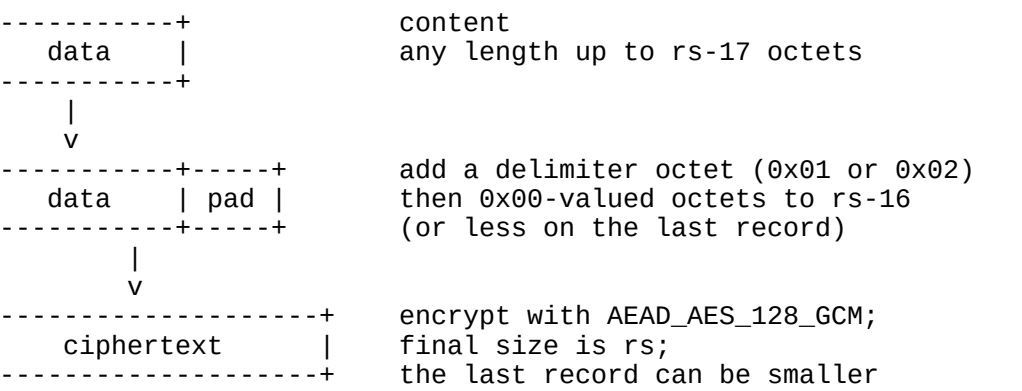
The "aes128gcm" HTTP content coding indicates that a payload has been encrypted using Advanced Encryption Standard (AES) in Galois/Counter Mode (GCM) as identified as AEAD_AES_128_GCM in [RFC5116], Section 5.1. The AEAD_AES_128_GCM algorithm uses a 128 bit content encryption key.

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

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

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

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



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

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

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

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

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

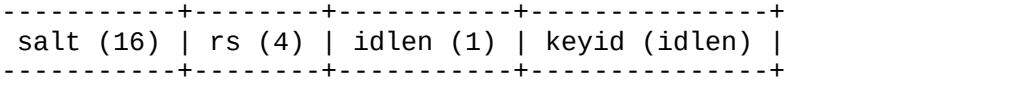
A consequence of this record structure is that range requests [RFC7233] and random access to encrypted payload bodies are possible at the granularity of the record size. Partial records at the ends of a range cannot be decrypted. Thus, it is best if range requests start and end on record boundaries.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

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

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

2.1. Encryption Content Coding Header

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



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

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

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

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

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

2.2. Content Encryption Key Derivation

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

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

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

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

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

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

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

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

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

2.3. Nonce Derivation

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

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

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

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

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

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

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

This nonce construction prevents removal or reordering of records.

3. Examples

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

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

3.1. Encryption of a Response

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

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

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

I1BsxtFttlv3u_Oo94xnmwAAEAAA-NAVub2qFgBEuQKRapoZu-IxkIva3MEB1PD-
ly8Thjg
```

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

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

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

3.2. Encryption with Multiple Records

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

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

uNCkWiNYzKTnBN9ji3-qWAAAABkCYTHOG8chz_gnvgOqdGYovxyjuqRyJFjEDyoF
1Fvkj6hQPdPHI51OEUKEpgz3SsLWIqS_uA
```

4. Security Considerations

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

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

4.1. Automatic Decryption

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

4.2. Message Truncation

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

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

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

4.3. Key and Nonce Reuse

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

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

4.4. Data Encryption Limits

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

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

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

4.5. Content Integrity

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

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

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

4.6. Leaking Information in Header Fields

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

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

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

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

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

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

4.7. Poisoning Storage

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

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

4.8. Sizing and Timing Attacks

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

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

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

5. IANA Considerations

5.1. The "aes128gcm" HTTP Content Coding

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

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

6. References

6.1. Normative References

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

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

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

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

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

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

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

6.2. Informative References

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

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

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

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

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

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

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

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

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

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

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

Appendix A. JWE Mapping

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

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

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

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

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

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

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

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

Appendix B. Acknowledgements

Mark Nottingham was an original author of this document.

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

Author’s Address

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

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

M. Nottingham

M. Thomson
Mozilla

March 17, 2017

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

Abstract

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

Status of This Memo

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

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

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

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

Table of Contents

1. Introduction

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

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

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

1.1. Goals and Non-Goals

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

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

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

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

1.2. Notational Conventions

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

2. Using HTTP URIs over TLS

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

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

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

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

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

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

2.1. Alternative Server Opt-In

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

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

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

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

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

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

2.2. Interaction with "https" URIs

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

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

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

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

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

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

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

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

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

3. IANA Considerations

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

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

4. Security Considerations

4.1. Security Indicators

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

4.2. Downgrade Attacks

A downgrade attack against the negotiation for TLS is possible.

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

4.3. Privacy Considerations

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

4.4. Confusion Regarding Request Scheme

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

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

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

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

4.5. Server Controls

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

5. References

5.1. Normative References

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

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

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

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

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

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

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

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

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

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

5.2. Informative References

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

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

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

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

Appendix A. Acknowledgements

Mike Bishop contributed significant text to this document.

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

Authors' Addresses

Mark Nottingham

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

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

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

R. Fielding
Adobe Systems Incorporated
M. Nottingham

March 2, 2016

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

Abstract

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

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

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

Note to Readers

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

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

Status of This Memo

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

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

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

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

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

Table of Contents

1. Introduction

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

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

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

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

1.1. Examples

For example, this response header field:

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

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

This Key:

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

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

A more complex example:

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

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

1.2. Notational Conventions

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

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

2. The "Key" Response Header Field

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

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

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

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

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

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

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

The following header fields have the same effect:

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

However, Key's use of parameters allows:

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

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

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

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

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

2.1. Relationship with Vary

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

For example,

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

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

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

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

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

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

2.2. Calculating a Secondary Cache Key

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

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

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

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

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

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

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

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

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

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

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

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

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

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

2.2.1. Creating a Header Field Value

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

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

2.2.2. Failing Parameter Processing

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

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

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

2.3. Key Parameters

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

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

2.3.1. div

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

Its value’s syntax is:

```
div    = 1*DIGIT
```

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

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

For example, the Key:

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

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

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

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

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

2.3.2. partition

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

Its value's syntax is:

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

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

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

For example, the Key:

Key: Foo;partition=20:30:40

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

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

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

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

whereas the following would fall into the second segment:

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

2.3.3. match

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

Its value's syntax is:

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

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

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

For example, the Key:

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

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

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

and "0" for these:

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

2.3.4. substr

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

Its value’s syntax is:

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

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

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

For example, the Key:

Key: Abc;substr=bennet

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

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

and "0" for these:

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

2.3.5. param

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

Its value's syntax is:

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

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

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

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

5) Return the empty string.

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

For example, the Key:

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

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

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

3. IANA Considerations

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

3.1. Procedure

Key Parameter registrations MUST include the following fields:

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

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

3.2. Registrations

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

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

4. Security Considerations

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

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

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

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

5. References

5.1. Normative References

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

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

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

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

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

5.2. Informative References

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

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

Appendix A. Acknowledgements

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

Appendix B. Changes

B.1. Since -00

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

Authors' Addresses

Roy T. Fielding
Adobe Systems Incorporated

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

Mark Nottingham

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

ï»¿

HTTP Working Group
Internet-Draft
Obsoletes: 5987 (if approved)
Intended status: Standards Track
Expires: September 3, 2017

J. Reschke
greenbytes

March 2, 2017

Indicating Character Encoding and Language for HTTP Header Field Parameters
draft-ietf-httpbis-rfc5987bis-05

Abstract

By default, header field values in Hypertext Transfer Protocol (HTTP) messages cannot easily carry characters outside the US-ASCII coded character set. RFC 2231 defines an encoding mechanism for use in parameters inside Multipurpose Internet Mail Extensions (MIME) header field values. This document specifies an encoding suitable for use in HTTP header fields that is compatible with a simplified profile of the encoding defined in RFC 2231.

This document obsoletes RFC 5987.

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

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

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

The changes in this draft are summarized in Appendix C.

Status of This Memo

material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 3, 2017.

Table of Contents

1. Introduction

Use of characters outside the US-ASCII coded character set ([RFC0020]) in HTTP header fields ([RFC7230]) is non-trivial:

- The HTTP specification discourages use of non-US-ASCII characters in field values, placing them into the "obs-text" ABNF production ([RFC7230], Section 3.2).

- Furthermore, it stays silent about default character encoding schemes for field values, so any use of non-US-ASCII characters would need to be specific to the field definition, or would require some other kind of out-of-band information.

- Finally, some APIs assume a default character encoding scheme in order to map from the octet sequences (obtained from the HTTP message) to character sequences: for instance, the XMLHttpRequest API ([XMLHttpRequest]) uses the Interface Definition Language type "ByteString", effectively resulting in the ISO-8859-1 character encoding scheme [ISO-8859-1] being used.

On the other hand, RFC 2231 defines an encoding mechanism for parameters inside MIME header fields ([RFC2231]), which, as opposed to HTTP messages, do need to be sent over non-binary transports. This document specifies an encoding suitable for use in HTTP header fields that is compatible with a simplified profile of the encoding defined in RFC 2231. It can be applied to any HTTP header field that uses the common "parameter" ("name=value") syntax.

This document obsoletes [RFC5987] and moves it to "historic" status; the changes are summarized in Appendix A.

> Note: in the remainder of this document, RFC 2231 is only referenced for the purpose of explaining the choice of features that were adopted; they are therefore purely informative.
>
> Note: this encoding does not apply to message payloads transmitted over HTTP, such as when using the media type "multipart/form-data" ([RFC7578]).

2. Notational Conventions

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

This specification uses the ABNF (Augmented Backus-Naur Form) notation defined in [RFC5234]. The following core rules are included

by reference, as defined in [RFC5234], Appendix B.1: ALPHA (letters), DIGIT (decimal 0-9), HEXDIG (hexadecimal 0-9/A-F/a-f), and LWSP (linear whitespace).

This specification uses terminology defined in [RFC6365], namely: "character encoding scheme" (below abbreviated to "character encoding"), "charset" and "coded character set".

Note that this differs from RFC 2231, which uses the term "character set" for "character encoding scheme".

3. Comparison to RFC 2231 and Definition of the Encoding

RFC 2231 defines several extensions to MIME. The sections below discuss if and how they apply to HTTP header fields.

In short:

- Parameter Continuations aren’t needed (Section 3.1),
- Character Encoding and Language Information are useful, therefore a simple subset is specified (Section 3.2), and
- Language Specifications in Encoded Words aren’t needed (Section 3.3).

3.1. Parameter Continuations

Section 3 of [RFC2231] defines a mechanism that deals with the length limitations that apply to MIME headers. These limitations do not apply to HTTP ([RFC7231], Appendix A.6).

Thus, parameter continuations are not part of the encoding defined by this specification.

3.2. Parameter Value Character Encoding and Language Information

Section 4 of [RFC2231] specifies how to embed language information into parameter values, and also how to encode non-ASCII characters, dealing with restrictions both in MIME and HTTP header field parameters.

However, RFC 2231 does not specify a mandatory-to-implement character encoding, making it hard for senders to decide which encoding to use. Thus, recipients implementing this specification MUST support the "UTF-8" character encoding [RFC3629].

Furthermore, RFC 2231 allows the character encoding information to be

left out. The encoding defined by this specification does not allow that.

3.2.1. Definition

The presence of extended parameter values usually is indicated by a parameter name ending in an asterisk character. Note however that this is just a convention, and that it needs to be explicitly specified in the definition of the header field using this extension (see Section 4).

The ABNF for extended parameter values is specified below:

```
ext-value     = charset  "'" [ language ] "'" value-chars
              ; like RFC 2231's <extended-initial-value>
              ; (see [RFC2231], Section 7)

charset       = "UTF-8" / mime-charset

mime-charset  = 1*mime-charsetc
mime-charsetc = ALPHA / DIGIT
              / "!" / "#" / "$" / "%" / "&"
              / "+" / "-" / "^" / "_" / "'"
              / "{" / "}" / "~"
              ; as <mime-charset> in Section 2.3 of [RFC2978]
              ; except that the single quote is not included
              ; SHOULD be registered in the IANA charset registry

language      = <Language-Tag, see [RFC5646], Section 2.1>

value-chars   = *( pct-encoded / attr-char )

pct-encoded   = "%" HEXDIG HEXDIG
              ; see [RFC3986], Section 2.1

attr-char     = ALPHA / DIGIT
              / "!" / "#" / "$" / "&" / "+" / "-" / "."
              / "^" / "_" / "'" / "|" / "~"
              ; token except ( "*" / "'" / "%" )
```

The value part of an extended parameter (ext-value) is a token that consists of three parts:

1. the REQUIRED character encoding name (charset),

2. the OPTIONAL language information (language), and

3. a character sequence representing the actual value (value-chars), separated by single quote characters.

Note that both character encoding names and language tags are restricted to the US-ASCII coded character set, and are matched case-insensitively (see [RFC2978], Section 2.3 and [RFC5646], Section 2.1.1).

Inside the value part, characters not contained in attr-char are encoded into an octet sequence using the specified character encoding. That octet sequence is then percent-encoded as specified in Section 2.1 of [RFC3986].

Producers MUST use the "UTF-8" ([RFC3629]) character encoding. Extension character encodings (mime-charset) are reserved for future use.

Note: recipients should be prepared to handle encoding errors, such as malformed or incomplete percent escape sequences, or non-decodable octet sequences, in a robust manner. This specification does not mandate any specific behavior, for instance, the following strategies are all acceptable:

* ignoring the parameter,

* stripping a non-decodable octet sequence,

* substituting a non-decodable octet sequence by a replacement character, such as the Unicode character U+FFFD (Replacement Character).

3.2.2. Historical Notes

The RFC 7230 token production ([RFC7230], Section 3.2.6) differs from the production used in RFC 2231 (imported from Section 5.1 of [RFC2045]) in that curly braces ("{" and "}") are excluded. Thus, these two characters are excluded from the attr-char production as well.

The <mime-charset> ABNF defined here differs from the one in Section 2.3 of [RFC2978] in that it does not allow the single quote character (see also RFC Errata ID 1912 [Err1912]). In practice, no character encoding names using that character have been registered at the time of this writing.

For backwards compatibility with RFC 2231, the encoding defined by this specification deviates from common parameter syntax in that the quoted-string notation is not allowed. Implementations using generic

parser components might not be able to detect the use of quoted-string notation and thus might accept that format, although invalid, as well.

[RFC5987] did require support for ISO-8859-1 ([ISO-8859-1]), too; for compatibility with legacy code, recipients are encouraged to support this encoding as well.

3.2.3. Examples

Non-extended notation, using "token":

```
foo: bar; title=Economy
```

Non-extended notation, using "quoted-string":

```
foo: bar; title="US-$ rates"
```

Extended notation, using the Unicode character U+00A3 ("Â£", POUND SIGN):

```
foo: bar; title*=utf-8'en'%C2%A3%20rates
```

Note: the Unicode pound sign character U+00A3 was encoded into the octet sequence C2 A3 using the UTF-8 character encoding, then percent-encoded. Also, note that the space character was encoded as %20, as it is not contained in attr-char.

Extended notation, using the Unicode characters U+00A3 ("Â£", POUND SIGN) and U+20AC ("â\202¬", EURO SIGN):

```
foo: bar; title*=UTF-8''%c2%a3%20and%20%e2%82%ac%20rates
```

Note: the Unicode pound sign character U+00A3 was encoded into the octet sequence C2 A3 using the UTF-8 character encoding, then percent-encoded. Likewise, the Unicode euro sign character U+20AC was encoded into the octet sequence E2 82 AC, then percent-encoded. Also note that HEXDIG allows both lowercase and uppercase characters, so recipients must understand both, and that the language information is optional, while the character encoding is not.

3.3. Language Specification in Encoded Words

Section 5 of [RFC2231] extends the encoding defined in [RFC2047] to also support language specification in encoded words. RFC 2616, the now-obsolete HTTP/1.1 specification, did refer to RFC 2047 ([RFC2616], Section 2.2). However, it wasn't clear to which header field it applied. Consequently, the current revision of the HTTP/1.1

specification has deprecated use of the encoding forms defined in RFC 2047 (see Section 3.2.4 of [RFC7230]).

Thus, this specification does not include this feature.

4. Guidelines for Usage in HTTP Header Field Definitions

Specifications of HTTP header fields that use the extensions defined in Section 3.2 ought to clearly state that. A simple way to achieve this is to normatively reference this specification, and to include the ext-value production into the ABNF for specific header field parameters.

For instance:

```
foo         = token ";" LWSP title-param
title-param = "title" LWSP "=" LWSP value
            / "title*" LWSP "=" LWSP ext-value
ext-value   = <see draft-ietf-httpbis-rfc5987bis, Section 3.2>
```

[[pub: Upon publication as RFC, the string "draft-ietf-httpbis-rfc5987bis" needs to be replaced with the RFC name, and this comment needs to be removed.]]

> Note: The Parameter Value Continuation feature defined in Section 3 of [RFC2231] makes it impossible to have multiple instances of extended parameters with identical names, as the processing of continuations would become ambiguous. Thus, specifications using this extension are advised to disallow this case for compatibility with RFC 2231.
>
> Note: This specification does not automatically assign a new interpretation to parameter names ending in an asterisk. As pointed out above, it’s up to the specification for the non-extended parameter to "opt in" to the syntax defined here. That being said, some existing implementations are known to automatically switch to the use of this notation when a parameter name ends with an asterisk, thus using parameter names ending in an asterisk for something else is likely to cause interoperability problems.

4.1. When to Use the Extension

Section 4.2 of [RFC2277] requires that protocol elements containing human-readable text are able to carry language information. Thus, the ext-value production ought to be always used when the parameter value is of textual nature and its language is known.

Furthermore, the extension ought to also be used whenever the parameter value needs to carry characters not present in the US-ASCII ([RFC0020]) coded character set (note that it would be unacceptable to define a new parameter that would be restricted to a subset of the Unicode character set).

4.2. Error Handling

Header field specifications need to define whether multiple instances of parameters with identical names are allowed, and how they should be processed. This specification suggests that a parameter using the extended syntax takes precedence. This would allow producers to use both formats without breaking recipients that do not understand the extended syntax yet.

Example:

```
  foo: bar; title="EURO exchange rates";
            title*=utf-8''%e2%82%ac%20exchange%20rates
```

In this case, the sender provides an ASCII version of the title for legacy recipients, but also includes an internationalized version for recipients understanding this specification -- the latter obviously ought to prefer the new syntax over the old one.

5. Security Considerations

The format described in this document makes it possible to transport non-ASCII characters, and thus enables character "spoofing" scenarios, in which a displayed value appears to be something other than it is.

Furthermore, there are known attack scenarios relating to decoding UTF-8.

See Section 10 of [RFC3629] for more information on both topics.

In addition, the extension specified in this document makes it possible to transport multiple language variants for a single parameter, and such use might allow spoofing attacks, where different language versions of the same parameter are not equivalent. Whether this attack is useful as an attack depends on the parameter specified.

6. IANA Considerations

There are no IANA Considerations related to this specification.

7. References

7.1. Normative References

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

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

[RFC2978] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, DOI 10.17487/RFC2978, October 2000, <http://www.rfc-editor.org/info/rfc2978>.

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

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

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

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <http://www.rfc-editor.org/info/rfc5646>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231,

June 2014,
<http://www.rfc-editor.org/info/rfc7231>.

7.2. Informative References

[Err1912] RFC Errata, "Errata ID 1912, RFC 2978", October 2009, <http://www.rfc-editor.org>.

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

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, DOI 10.17487/RFC2045, November 1996, <http://www.rfc-editor.org/info/rfc2045>.

[RFC2047] Moore, K., "MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text", RFC 2047, DOI 10.17487/RFC2047, November 1996, <http://www.rfc-editor.org/info/rfc2047>.

[RFC2231] Freed, N. and K. Moore, "MIME Parameter Value and Encoded Word Extensions: Character Sets, Languages, and Continuations", RFC 2231, DOI 10.17487/RFC2231, November 1997, <http://www.rfc-editor.org/info/rfc2231>.

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, DOI 10.17487/RFC2277, January 1998, <http://www.rfc-editor.org/info/rfc2277>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

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

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

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

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

[RFC7578] Masinter, L., "Returning Values from Forms: multipart/form-data", RFC 7578, DOI 10.17487/RFC7578, July 2015, <http://www.rfc-editor.org/info/rfc7578>.

[RFC7616] Shekh-Yusef, R., Ed., Ahrens, D., and S. Bremer, "HTTP Digest Access Authentication", RFC 7616, DOI 10.17487/RFC7616, September 2015, <http://www.rfc-editor.org/info/rfc7616>.

[RFC8053] Oiwa, Y., Watanabe, H., Takagi, H., Maeda, K., Hayashi, T., and Y. Ioku, "HTTP Authentication Extensions for Interactive Clients", RFC 8053, DOI 10.17487/RFC8053, January 2017, <http://www.rfc-editor.org/info/rfc8053>.

[XMLHttpRequest] WhatWG, "XMLHttpRequest", <https://xhr.spec.whatwg.org/>.

Appendix A. Changes from RFC 5987

This section summarizes the changes compared to [RFC5987]:

- The document title was changed to "Indicating Character Encoding and Language for HTTP Header Field Parameters".
- The introduction was rewritten to better explain the issues around non-ASCII characters in field values.
- The requirement to support the "ISO-8859-1" encoding was removed.
- The document does not attempt to re-define a generic "parameter" ABNF anymore (it turned out that there really isn’t a generic definition of parameters in HTTP; for instance, there are subtle

differences with respect to whitespace handling).

- A note about defects in error handling in current implementations was removed, as it wasn't accurate anymore.

Appendix B. Implementation Report

The encoding defined in this document currently is used in four different HTTP header fields:

- "Authentication-Control", defined in [RFC8053],
- "Authorization" (as used in HTTP Digest Authentication, defined in [RFC7616]),
- "Content-Disposition", defined in [RFC6266], and
- "Link", defined in [RFC5988].

As the encoding is a profile/clarification of the one defined in [RFC2231] in 1997, many user agents already supported it for use in "Content-Disposition" when [RFC5987] got published.

Since the publication of [RFC5987], three more popular desktop user agents have added support for this encoding; see <http://purl.org/NET/http/content-disposition-tests#encoding-2231-char> for details. At this time, the current versions of all major desktop user agents support it.

Note that the implementation in Internet Explorer 9 does not support the ISO-8859-1 character encoding; this document revision acknowledges that UTF-8 is sufficient for expressing all code points, and removes the requirement to support ISO-8859-1.

The "Link" header field, on the other hand, was more recently specified in [RFC5988]. At the time of this writing, no User Agent except Firefox supported the "title*" parameter (starting with release 15).

Section 3.4 of [RFC7616] defines the "username*" parameter for use in HTTP Digest Authentication. At the time of writing, no User Agent implemented this extension.

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

C.1. Since RFC5987

Only editorial changes for the purpose of starting the revision process (obs5987).

C.2. Since draft-reschke-rfc5987bis-00

Resolved issues "iso-8859-1" and "title" (title simplified). Added and resolved issue "historic5987".

C.3. Since draft-reschke-rfc5987bis-01

Added issues "httpbis", "parmsyntax", "terminology" and "valuesyntax". Closed issue "impls".

C.4. Since draft-reschke-rfc5987bis-02

Resolved issue "terminology".

C.5. Since draft-reschke-rfc5987bis-03

In Section 3.2, pull historical notes into a separate subsection. Resolved issues "valuesyntax" and "parmsyntax".

C.6. Since draft-reschke-rfc5987bis-04

Update status of Firefox support in HTTP Link Header field.

C.7. Since draft-reschke-rfc5987bis-05

Update status of Firefox support in HTTP Link Header field.

C.8. Since draft-reschke-rfc5987bis-06

Update status with respect to Safari 6.

Started work on update with respect to RFC 723x.

C.9. Since draft-ietf-httpbis-rfc5987bis-00

Editorial changes; introducing non-ASCII characters into author’s address, acknowledgements, and examples.

C.10. Since draft-ietf-httpbis-rfc5987bis-01

Removed mention of RFC 2616 from Abstract and Introduction.

Reference RFC 20 for US-ASCII.

Do not attempt to define a generic parameter ABNF; just concentrate on the parameter value syntax.

C.11. Since draft-ietf-httpbis-rfc5987bis-02

RFC 2388 -> RFC 7578.

Expand on the motivation (see <https://github.com/httpwg/http-extensions/issues/213>).

Mention RFC 7616 in implementation report.

C.12. Since draft-ietf-httpbis-rfc5987bis-03

Fixed one editorial issue. Updated XHR reference.

Fixed <https://github.com/httpwg/http-extensions/issues/266>: use of now undefined term "parmname".

Include WG into Acknowledgements for this revision.

Mention RFC 5987 in the abstract (<https://github.com/httpwg/http-extensions/issues/271>).

C.13. Since draft-ietf-httpbis-rfc5987bis-04

Mention RFC8053 in Implementation Report.

Appendix D. Acknowledgements

Thanks to Martin DÃ¼rst and Frank Ellermann for help figuring out ABNF details, to Graham Klyne and Alexey Melnikov for general review, to Chris Newman for pointing out an RFC 2231 incompatibility, and to Benjamin Carlyle, Roar Lauritzsen, Eric Lawrence, and James Manger for implementer’s feedback.

Furthermore thanks to the members of the IETF HTTP Working Group for the feedback specific to this update of RFC 5987.

Author's Address

Julian F. Reschke
greenbytes GmbH
Hafenweg 16
MÃ¼nster, NW 48155
Germany

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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: December 10, 2016

K. Oku
DeNA Co, Ltd.
M. Nottingham

June 8, 2016

Cache Digests for HTTP/2
draft-kazuho-h2-cache-digest-01

Abstract

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

Note to Readers

The issues list for this draft can be found at https://github.com/mnot/I-D/labels/h2-cache-digest .

The most recent (often, unpublished) draft is at https://mnot.github.io/I-D/h2-cache-digest/ .

Recent changes are listed at https://github.com/mnot/I-D/commits/gh-pages/h2-cache-digest .

Status of This Memo

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

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

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

This Internet-Draft will expire on December 10, 2016.

Table of Contents

1. Introduction

HTTP/2 [RFC7540] allows a server to "push" synthetic request/response pairs into a client's cache optimistically. While there is strong interest in using this facility to improve perceived Web browsing performance, it is sometimes counterproductive because the client might already have cached the "pushed" response.

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

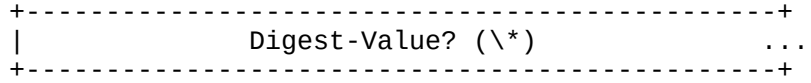
This specification defines a HTTP/2 frame type to allow clients to inform the server of their cache's contents using a Golumb-Rice Coded Set. Servers can then use this to inform their choices of what to push to clients.

1.1. Notational Conventions

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

2. The CACHE_DIGEST Frame

The CACHE_DIGEST frame type is 0xf1. NOTE: This is an experimental value; if standardised, a permanent value will be assigned.



The CACHE_DIGEST frame payload has the following fields:

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

The CACHE_DIGEST frame defines the following flags:

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

- *COMPLETE* (0x2): When set, indicates that the currently valid set of cache digests held by the server constitutes a complete representation of the cache’s state regarding that origin, for the type of cached response indicated by the "STALE" flag.

- *VALIDATORS* (0x4): When set, indicates that the "validators" boolean in Section 2.1.1 is true.

- *STALE* (0x8): When set, indicates that all cached responses represented in the digest-value are stale [RFC7234] at the point in them that the digest was generated; otherwise, all are fresh.

2.1. Client Behavior

A CACHE_DIGEST frame can be sent from a client to a server on any stream in the "open" state, and conveys a digest of the contents of the client’s cache for associated stream.

In typical use, a client will send one or more CACHE_DIGESTs immediately after the first request on a connection for a given origin, on the same stream, because there is usually a short period of inactivity then, and servers can benefit most when they understand

the state of the cache before they begin pushing associated assets (e.g., CSS, JavaScript and images). Clients MAY send CACHE_DIGEST at other times.

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

When generating CACHE_DIGEST, a client MUST NOT include cached responses whose URLs do not share origins [RFC6454] with the request of the stream that the frame is sent upon.

CACHE_DIGEST allows the client to indicate whether the set of URLs used to compute the digest represent fresh or stale stored responses, using the STALE flag. Clients MAY decide whether to only sent CACHE_DIGEST frames representing their fresh stored responses, their stale stored responses, or both.

Clients can choose to only send a subset of the suitable stored responses of each type (fresh or stale). However, when the CACHE_DIGEST frames sent represent the complete set of stored responses of a given type, the last such frame SHOULD have a COMPLETE flag set, to indicate to the server that it has all relevant state of that type. Note that for the purposes of COMPLETE, responses cached since the beginning of the connection or the last RESET flag on a CACHE_DIGEST frame need not be included.

CACHE_DIGEST can be computed to include cached responses' ETags, as indicated by the VALIDATORS flag. This information can be used by servers to decide what kinds of responses to push to clients; for example, a stale response that hasn't changed could be refreshed with a 304 (Not Modified) response; one that has changed can be replaced with a 200 (OK) response, whether the cached response was fresh or stale.

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

2.1.1. Computing the Digest-Value

Given the following inputs:

- "validators", a boolean indicating whether validators ([RFC7232]) are to be included in the digest;

- "URLs'", an array of (string "URL", string "ETag") tuples, each corresponding to the Effective Request URI ([RFC7230], Section 5.5) of a cached response [RFC7234] and its entity-tag

[RFC7232] (if "validators" is true and if the ETag is available; otherwise, null);

- "P", an integer that MUST be a power of 2 smaller than 2**32, that indicates the probability of a false positive that is acceptable, expressed as "1/P".

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

1. Let N be the count of "URLs"' members, rounded to the nearest power of 2 smaller than 2**32.
2. Let "hash-values" be an empty array of integers.
3. Append 0 to "hash-values".
4. For each ("URL", "ETag") in "URLs", compute a hash value (Section 2.1.2) and append the result to "hash-values".
5. Sort "hash-values" in ascending order.
6. Let "digest-value" be an empty array of bits.
7. Write log base 2 of "N" to "digest-value" using 5 bits.
8. Write log base 2 of "P" to "digest-value" using 5 bits.
9. For each "V" in "hash-values":
    1. Let "W" be the value following "V" in "hash-values".
    2. If "W" and "V" are equal, continue to the next "V".
    3. Let "D" be the result of "W - V - 1".
    4. Let "Q" be the integer result of "D / P".
    5. Let "R" be the result of "D modulo P".
    6. Write "Q" '0' bits to "digest-value".
    7. Write 1 '1' bit to "digest-value".
    8. Write "R" to "digest-value" as binary, using log2("P"5) bits.

9. If "V" is the second-to-last member of "hash-values", stop iterating through "hash-values" and continue to the next step.

10. If the length of "digest-value" is not a multiple of 8, pad it with 0s until it is.

2.1.2. Computing a Hash Value

Given:

- "URL", an array of characters
- "ETag", an array of characters
- "validators", a boolean
- "N", an integer
- "P", an integer

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

1. Let "key" be "URL" converted to an ASCII string by percent-encoding as appropriate [RFC3986].
2. If "validators" is true and "ETag" is not null:
    1. Append "ETag" to "key" as an ASCII string, including both the "weak" indicator (if present) and double quotes, as per [RFC7232] Section 2.3.
3. Let "hash-value" be the SHA-256 message digest [RFC6234] of "key", expressed as an integer.
4. Truncate "hash-value" to log2("N" * "P") bits.

2.2. Server Behavior

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

- If a given URL has a match in a current CACHE_DIGEST with the STALE flag unset, it need not be pushed, because it is fresh in cache;

- If a given URL and ETag combination has a match in a current CACHE_DIGEST with the STALE flag set, the client has a stale copy in cache, and a validating response can be pushed;

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

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

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

2.2.1. Querying the Digest for a Value

Given:

- "digest-value", an array of bits

- "URL", an array of characters

- "ETag", an array of characters

- "validators", a boolean

we can determine whether there is a match in the digest using the following algorithm:

1. Read the first 5 bits of "digest-value" as an integer; let "N" be two raised to the power of that value.

2. Read the next 5 bits of "digest-value" as an integer; let "P" be two raised to the power of that value.

3. Let "hash-value" be the result of computing a hash value (Section 2.1.2).

4. Let "C" be -1.

5. Read ’0’ bits from "digest-value" until a ’1’ bit is found; let "Q" bit the number of ’0’ bits. Discard the ’1’.

6. Read log2("P") bits from "digest-value" after the ’1’ as an integer; let "R" be its value.

7. Let "D" be "Q" * "P" + "R".

8. Increment "C" by "D" + 1.

9. If "C" is equal to "hash-value", return ’true’.

10. Otherwise, return to step 5 and continue processing; if no match is found before "digest-value" is exhausted, return ’false’.

3. IANA Considerations

This draft currently has no requirements for IANA. If the CACHE_DIGEST frame is standardised, it will need to be assigned a frame type.

4. Security Considerations

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

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

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

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

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

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>.

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

[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>.

[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>.

5.2. Informative References

[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 Adam Langley and Giovanni Bajo for their explorations of Golumb-coded sets. In particular, see http://giovanni.bajo.it/post/47119962313/golomb-coded-sets-smaller-than-bloom-filters , which refers to sample code.

Thanks to Stefan Eissing for his suggestions.

Authors' Addresses

Kazuho Oku
DeNA Co, Ltd.

Email: kazuhooku@gmail.com

Mark Nottingham

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

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.

Table of Contents

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.

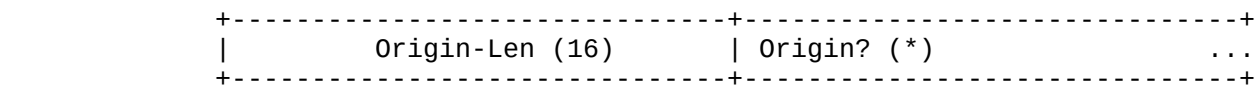
It MUST occur on stream 0; an ORIGIN frame on any other stream is invalid and MUST be ignored.

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).



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: Standards Track
Expires: January 3, 2019

J. Reschke
greenbytes

July 2, 2018

A JSON Encoding for HTTP Header Field Values
draft-reschke-http-jfv-09

Abstract

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

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

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 [1], which may be joined by sending a message with subject "subscribe" to ietf-http-wg-request@w3.org [2].

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 E.12.

Status of This Memo

Table of Contents

1. Introduction

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

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

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

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

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

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

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

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

  Note: [HSTRUCT], a work item of the IETF HTTP Working Group, is a different attempt to address this set of problems -- it tries to identify and formalize common field structures in existing header fields; the syntax defined over there would usually lead to a more compact notation.

2. Data Model and Format

In HTTP, header fields with the same field name can occur multiple times within a single message (Section 3.2.2 of [RFC7230]). When this happens, recipients are allowed to combine the field values using commas as delimiter. This rule matches nicely JSON's array format (Section 5 of [RFC8259]). Thus, the basic data model used here is the JSON array.

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

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

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

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

Characters in JSON strings that are not allowed or discouraged in HTTP header field values -- that is, not in the "VCHAR" definition -- need to be represented using JSON's "backslash" escaping mechanism ([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 7 of [RFC7230]:

```
  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 header field value, process each array element separately by:

1. generating the JSON representation,

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

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

4. Recipient Requirements

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

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

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

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

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

5. Using this Format in Header Field Definitions

Specifications defining new HTTP header fields need to take the considerations listed in Section 8.3.1 of [RFC7231] 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 header fields, thus backwards compatibility is not a problem. That being said, it is conceivable that there is existing code that might trip over double quotes not being used for HTTP's quoted-string syntax (Section 3.2.6 of [RFC7230]).

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 use a subset of UTF-8 (Section 2.1 of [RFC7493]).

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 MUST NOT use duplicate object names, and recipients SHOULD either treat field values with duplicate names as invalid (consistent with [RFC7493], Section 2.3) or use the lexically last value (consistent with [ECMA-262], Section 24.3.1.1).

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

8. Internationalization Considerations

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

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

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

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

[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>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

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

[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>.

10.2. Informative References

[CLEARSITE] West, M., "Clear Site Data", W3C Working Draft WD-clear-site-data-20171130, 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/>.

[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/>.

[FEATUREPOL] Clelland, I., "Feature Policy", W3C Draft Community Group Report , June 2018, <https://wicg.github.io/feature-policy/>.

[HSTRUCT] Nottingham, M. and P-H. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-07 (work in progress), July 2018.

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

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

[REPORTING]
Grigorik, I. and M. West, "Reporting API 1", W3C Group Note NOTE-reporting-1-20160607, June 2016, <http://www.w3.org/TR/2016/NOTE-reporting-1-20160607/>.

Latest version available at <http://www.w3.org/TR/reporting-1/>.

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

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

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

[RFC8187] Reschke, J., "Indicating Character Encoding and Language for HTTP Header Field Parameters", RFC 8187, DOI 10.17487/RFC8187, September 2017, <https://www.rfc-editor.org/info/rfc8187>.

[XMLHttpRequest]
WhatWG, "XMLHttpRequest", <https://xhr.spec.whatwg.org/>.

10.3. URIs

[1] mailto:ietf-http-wg@w3.org

[2] mailto:ietf-http-wg-request@w3.org?subject=subscribe

Appendix A. Examples

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

A.1. Content-Length

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

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

So the field value is similar to a JSON number ([RFC8259], Section 6).

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

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

```
Content-Length = #number
                 ; number: [RFC8259], Section 6
```

...and the prose definition would:

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

A.2. Content-Disposition

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

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

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

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

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

which would translate to a header field value of:

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

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

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

Note that in this case, the "filename*" parameter uses the encoding defined in [RFC8187], representing a filename starting with the Unicode character U+20AC (EURO SIGN), followed by " rates". If the definition of Content-Disposition would have used the format proposed here, the workaround involving the "parameter*" syntax would not have been needed at all.

The JSON representation of this value could then be:

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

A.3. WWW-Authenticate

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

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

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

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

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

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

(line break added for readability)

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

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

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

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

A.4. Accept-Encoding

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

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

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

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

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

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

A JSON representation could be:

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

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

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

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

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

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

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

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

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

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

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 [HSTRUCT] (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 Note "Reporting API 1" (Section 3.1 of [REPORTING]). Still in use in latest editor copy as of June 2017.

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 Draft Community Group Report "Feature Policy" ([FEATUREPOL]), but now replaced with a custom syntax (see <https://github.com/WICG/feature-policy/pull/83>).

Appendix C. Relation to HTTP 'Key' Header Field

[KEY] aims to improve the cacheability of responses that vary based on certain request header fields, addressing lack of granularity in the existing "Vary" response header field ([RFC7231], Section 7.1.4). If the JSON-based format described by this document gains popularity, it might be useful to add a JSON-aware "Key Parameter" (see Section 2.3 of [KEY]).

Appendix D. Discussion

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

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

[[CREF1: Use of generic libs vs compactness of field values..]]

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

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

Editorial fixes + working on the TODOs.

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

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

E.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.

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

Mention relation to KEY header field.

E.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:

E.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.

E.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>).

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

Move non-essential parts into appendix.

Updated XHR reference.

E.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.

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

RFC 5987 is obsoleted by RFC 8187.

Update CLEARSITE comment.

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

Update JSON and HSTRUCT references.

FEATUREPOL doesn’t use JSON syntax anymore.

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

Update HSTRUCT reference.

Update notes about CLEARSITE and FEATUREPOL.

Acknowledgements

Thanks go to the Hypertext Transfer Protocol Working Group participants.

Author’s Address

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

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

httpbis
Internet-Draft
Intended status: Best Current Practice
Expires: May 4, 2017

D. Stenberg
Mozilla
T. Wicinski
Salesforce

October 31, 2016

TCP Tuning for HTTP
draft-stenberg-httpbis-tcp-03

Abstract

This document records current best practice for using all versions of HTTP over TCP.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 4, 2017.

Table of Contents

1. Introduction

HTTP version 1.1 [RFC7230] as well as HTTP version 2 [RFC7540] are defined to use TCP [RFC0793], and their performance can depend greatly upon how TCP is configured. This document records the best

current practice for using HTTP over TCP, with a focus on improving end-user perceived performance.

These practices are generally applicable to HTTP/1 as well as HTTP/2, although some may note particular impact or nuance regarding a particular protocol version.

There are countless scenarios, roles and setups where HTTP is being using so there can be no single specific "Right Answer" to most TCP questions. This document intends only to cover the most important areas of concern and suggest possible actions.

1.1. Notational Conventions

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

2. Socket planning

Your HTTP server or intermediary may need configuration changes to some system tunables and timeout periods to perform optimally. Actual values will depend on how you are scaling the platform, horizontally or vertically, and other connection semantics. Changing system limits and altering thresholds will change the behavior of your web service and its dependencies. These dependencies are usually common to other services running on the same system, so good planning and testing is advised.

This is a list of values to consider and some general advice on how those values can be modified on Linux systems.

2.1. Number of open files

A modern HTTP server will serve a large number of TCP connections and in most systems each open socket equals an open file. Make sure that limit isn’t a bottle neck.

2.2. Number of concurrent network messages

Raise the number of packets allowed to get queued when a particular interface receives packets faster than the kernel can process them.

2.3. Number of incoming TCP SYNs allowed to backlog

The number of new connection requests that are allowed to queue up in the kernel. These can be connections that are in SYN RECEIVED or ESTABLISHED states. Historically, operating systems used a single

backlog queue for both of these states. Newer implemntations use two separate queues: one for connections in SYN RECEIVED and one for those which are ESTABLISHED state (better known as the accept queue).

2.4. Use the whole port range for local ports

To make sure the TCP stack can take full advantage of the entire set of possible sockets, give it a larger range of local port numbers to use.

2.5. Lower the TCP FIN timeout

High connection completion rates will consume ephemeral ports quickly. Lower the time during which connections are in FIN-WAIT-2/ TIME_WAIT states so that they can be purged faster and thus maintain a maximal number of available sockets. The primitives for the assignment of these values were described in [RFC0793], however significantly lower values are commonly used.

2.6. Reuse sockets in TIME_WAIT state

When running backend servers on a managed, low latency network you might allow the reuse of sockets in TIME_WAIT state for new connections when a protocol complete termination has occurred. There is no RFC that covers this behaviour.

2.7. TCP socket buffer sizes and Window Scaling

Systems meant to handle and serve a huge number of TCP connections at high speeds need a significant amount of memory for TCP socket buffers. On some systems you can tell the TCP stack what default buffer sizes to use and how much they are allowed to dynamically grow and shrink. Window Scaling is typically linked to socket buffer sizes.

The minimum and default tend to require less proactive amendment than the maximum value. When deriving maximum values for use, you should consider the BDP (Bandwidth Delay Product) of the target environment and clients. Consider also that ’read’ and ’write’ values do not require to be synchronised, as the BDP requirements for a load balancer or middle-box might be very different when acting as a sender or receiver.

Allowing needlessly high values beyond the expected limitations of the platform might increase the probability of retransmissions and buffer induced delays within the path. Extensions such as ECN coupled with AQM can help mitigate this undesirable behaviour [RFC7141].

[RFC7323] covers Window Scaling in greater detail.

2.8. Set maximum allowed TCP window sizes

You may have to increase the largest allowed window size. Window scaling must be accommodated within the maximal values, however it is not uncommon to see the maximum definable higher than the scalable limit; these values can statically defined within socket parameters (SO_RCVBUF,SO_SNDBUF).

2.9. Timers and timeouts

On a modern shared platform it can be common to plan for both long and short lived connections on the same implementation. However, the delivery of static assets and a 'web push' or 'long poll' service provide very different quality of service promises.

Fail 'fast': TCP resources can be highly contended. For fault tolerance reasons a server needs to be able to determine within a reasonable time frame whether a connection is still active or required. e.g. If static assets typically return in 100s of milliseconds, and users 'switch off' after <10s keeping timeouts of >30s make little sense and defining a 'quality of service' appropriate to the target platform is encouraged. On a shared platform with mixed session lifetimes, applications that require longer render times have various options to ensure the underlying service and upstream servers in the path can identify the session as not failed: HTTP continuations, Redirects, 202s or sending data.

Clients and servers typically have many timeout options, a few notable options are: Connect(client), time to request(server), time to first byte(client), between bytes(server/client), total connection time(server/client). Some implementations merge these values into a single 'timeout' definition even when statistics are reported individually. All should be considered as the defaults in many implementations are highly underiable, even infinite timeouts have been observed.

3. TCP handshake

3.1. TCP Fast Open

TCP Fast Open (a.k.a. TFO, [RFC7413]) allows data to be sent on the TCP handshake, thereby allowing a request to be sent without any delay if a connection is not open.

TFO requires both client and server support, and additionally requires application knowledge, because the data sent on the SYN

needs to be idempotent. Therefore, TFO can only be used on idempotent, safe HTTP methods (e.g., GET and HEAD), or with intervening negotiation (e.g, using TLS). It should be noted that TFO requires a secret to be defined on the server to mitigate security vulnerabilities it introduces. TFO therefore requires more server side deployment planning than other enhancements.

Support for TFO is growing in client platforms, especially mobile, due to the significant performance advantage it gives.

3.2. Initial Congestion Window

[RFC6928] specifies an initcwnd (initial congestion window) of 10, and is now fairly widely deployed server-side. There has been experimentation with larger initial windows, in combination with packet pacing. Many implementations allow initcwnd to be applied to specific routes which allows a greater degree of flexibility than some other TCP parameters.

IW10 has been reported to perform fairly well even in high volume servers.

3.3. TCP SYN flood handling

TCP SYN Flood mitigations [RFC4987] are necessary and there will be thresholds to tweak.

4. TCP transfers

4.1. Packet Pacing

TBD

4.2. Explicit Congestion Control

Apple deploying in iOS and OSX [1].

4.3. Nagle’s Algorithm

Nagle’s Algorithm [RFC0896] is the mechanism that makes the TCP stack hold (small) outgoing packets for a short period of time so that it can potentially merge that packet with the next outgoing one. It is optimized for throughput at the expense of latency.

HTTP/2 in particular requires that the client can send a packet back fast even during transfers that are perceived as single direction transfers. Even small delays in those sends can cause a significant performance loss.

HTTP/1.1 is also affected, especially when sending off a full request in a single write() system call.

In POSIX systems you switch it off like this:

```
int one = 1;
setsockopt(fd, IPPROTO_TCP, TCP_NODELAY, &one, sizeof(one));
```

4.4. Delayed ACKs

Delayed ACK [RFC1122] is a mechanism enabled in most TCP stacks that causes the stack to delay sending acknowledgement packets in response to data. The ACK is delayed up until a certain threshold, or until the peer has some data to send, in which case the ACK will be sent along with that data. Depending on the traffic flow and TCP stack this delay can be as long as 500ms.

This interacts poorly with peers that have Nagle's Algorithm enabled. Because Nagle's Algorithm delays sending until either one MSS of data is provided _or_ until an ACK is received for all sent data, delaying ACKs can force Nagle's Algorithm to buffer packets when it doesn't need to (that is, when the other peer has already processed the outstanding data).

Delayed ACKs can be useful in situations where it is reasonable to assume that a data packet will almost immediately (within 500ms) cause data to be sent in the other direction. In general in both HTTP/1.1 and HTTP/2 this is unlikely: therefore, disabling Delayed ACKs can provide an improvement in latency.

However, the TLS handshake is a clear exception to this case. For the duration of the TLS handshake it is likely to be useful to keep Delayed ACKs enabled.

Additionally, for low-latency servers that can guarantee responses to requests within 500ms, on long-running connections (such as HTTP/2), and when requests are small enough to fit within a small packet, leaving delayed ACKs turned on may provide minor performance benefits.

Effective use of switching off delayed ACKs requires extensive profiling.

4.5. Keep-alive

TCP keep-alive is likely disabled - at least on mobile clients for energy saving purposes. App-level keep-alive is then required for

long-lived requests to detect failed peers or connections reset by stateful firewalls etc.

5. Re-using connections

5.1. Slow Start after Idle

Slow-start is one of the algorithms that TCP uses to control congestion inside the network. It is also known as the exponential growth phase. Each TCP connection will start off in slow-start but will also go back to slow-start after a certain amount of idle time.

5.2. TCP-Bound Authentications

There are several HTTP authentication mechanisms in use today that are used or can be used to authenticate a connection rather than a single HTTP request. Two popular ones are NTLM and Negotiate.

If such an authentication has been negotiated on a TCP connection, that connection can remain authenticated throughout the rest of its lifetime. This discrepancy with how other HTTP authentications work makes it important to handle these connections with care.

6. Closing connections

6.1. Half-close

The client or server is free to half-close after a request or response has been completed; or when there is no pending stream in HTTP/2.

Half-closing is sometimes the only way for a server to make sure it closes down connections cleanly so that it doesn’t accept more requests while still allowing clients to receive the ongoing responses.

6.2. Abort

No client abort for HTTP/1.1 after the request body has been sent. Delayed full close is expected following an error response to avoid RST on the client.

6.3. Close Idle Connections

Keeping open connections around for subsequent connection reuse is key for many HTTP clients’ performance. The value of an existing connection quickly degrades and after only a few minutes the chance

that a connection will successfully get reused by a web browser is slim.

6.4. Tail Loss Probes

draft [2]

7. IANA Considerations

This document does not require action from IANA.

8. Security Considerations

TBD

9. References

9.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

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

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

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

9.2. Informative References

[RFC0896] Nagle, J., "Congestion Control in IP/TCP Internetworks", RFC 896, DOI 10.17487/RFC0896, January 1984, <http://www.rfc-editor.org/info/rfc896>.

[RFC1122] Braden, R., Ed., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, DOI 10.17487/RFC1122, October 1989, <http://www.rfc-editor.org/info/rfc1122>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <http://www.rfc-editor.org/info/rfc4987>.

[RFC6928] Chu, J., Dukkipati, N., Cheng, Y., and M. Mathis, "Increasing TCP's Initial Window", RFC 6928, DOI 10.17487/RFC6928, April 2013, <http://www.rfc-editor.org/info/rfc6928>.

[RFC7141] Briscoe, B. and J. Manner, "Byte and Packet Congestion Notification", BCP 41, RFC 7141, DOI 10.17487/RFC7141, February 2014, <http://www.rfc-editor.org/info/rfc7141>.

[RFC7323] Borman, D., Braden, B., Jacobson, V., and R. Scheffenegger, Ed., "TCP Extensions for High Performance", RFC 7323, DOI 10.17487/RFC7323, September 2014, <http://www.rfc-editor.org/info/rfc7323>.

[RFC7413] Cheng, Y., Chu, J., Radhakrishnan, S., and A. Jain, "TCP Fast Open", RFC 7413, DOI 10.17487/RFC7413, December 2014, <http://www.rfc-editor.org/info/rfc7413>.

9.3. URIs

[1] https://developer.apple.com/videos/wwdc/2015/?id=719

[2] http://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe-01

Appendix A. Acknowledgments

This specification builds upon previous work and help from Mark Nottingham, Craig Taylor

Appendix B. Operating System Settings for Linux

Here are some sample operating system settings for the Linux operating system, along with the section it refers to.

Section 2.1

fs.file-max = <number of files>

Section 2.2

net.core.netdev_max_backlog = <number of packets>

Section 2.3

net.core.somaxconn = <number>

Section 2.4

net.ipv4.ip_local_port_range = 1024 65535

Section 2.5

net.ipv4.tcp_fin_timeout = <number of seconds>

Section 2.6

net.ipv4.tcp_tw_reuse = 1

Section 2.7

net.ipv4.tcp_wmem = <minimum size> <default size> <max size in bytes>

Section 2.7

net.ipv4.tcp_rmem = <minimum size> <default size> <max size in bytes>

Section 2.8

net.core.rmem_max = <number of bytes>

Section 2.8

net.core.wmem_max = <number of bytes>

Section 5.1

net.ipv4.tcp_slow_start_after_idle = 0

Section 4.3 Turning off Nagle’s Algorithm:

```
int one = 1;
setsockopt(fd, IPPROTO_TCP, TCP_NODELAY, &one, sizeof(one));
```

Section 4.4

On recent Linux kernels (since Linux 2.4.4), Delayed ACKs can be disabled like this:

```
int one = 1;
setsockopt(fd, IPPROTO_TCP, TCP_QUICKACK, &one, sizeof(one));
```

Unlike disabling Nagle's Algorithm, disabling Delayed ACKs on Linux is not a one-time operation: processing within the TCP stack can cause Delayed ACKs to be re-enabled. As a result, to use "TCP_QUICKACK" effectively requires setting and unsetting the socket option during the life of the connection.

Authors' Addresses

Daniel Stenberg
Mozilla

Email: daniel@haxx.se
URI: http://daniel.haxx.se

Tim Wicinski
Salesforce

Email: tjw.ietf@gmail.com

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

M. Thomson
Mozilla
G. Eriksson
C. Holmberg
Ericsson

October 30, 2016

Caching Secure HTTP Content using Blind Caches
draft-thomson-http-bc-01

Abstract

A mechanism is described whereby a server can use client-selected shared cache.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2017.

Copyright Notice

Table of Contents

1. Shared Caching for HTTPS

Shared caches allow an HTTP server to offload the responsibility for delivering certain content. Content in the shared cache can be accessed efficiently by multiple clients, saving the origin server from having to serve those requests and ensuring that clients receive responses to cached requests more quickly.

Proxy caching is the most common configuration for shared caching. A proxy cache is either explicitly configured by a client, discovered as a result of being automatically configured.

HTTPS [RFC2818] prevents the use of proxies by creating an authenticated end-to-end connection to the origin server or its gateway that is authenticated. This provides a critical protection against man-in-the-middle attacks, but it also prevents the proxy from acting as a shared cache.

Clients do not direct queries for "https" URIs to proxies. Clients configured with a proxy use the CONNECT pseudo-method (Section 4.3.6 of [RFC7231]) with any explicitly configured or discovered proxies to create an end-to-end tunnel. Transparent proxies are unable to intercept connections that are protected with TLS.

This document describes a method that enables shared caching for a limited set of "https" resources, as selected by the server. The server conditionally delegates the hosting of secure content to itself. This delegation includes a marker that signals permission for a client to send a request for an "https" resource via a proxy rather than insisting on an end-to-end TLS connection.

1.1. Notational Conventions

The words "MUST", "MUST NOT", "SHOULD", and "MAY" are used in this document. It's not shouting; when they are capitalized, they have the special meaning defined in [RFC2119].

This document uses the term "proxy cache" to refer to a proxy [RFC7230] that operates an HTTP cache [RFC7234].

2. Same-Host Secure Content Delegation

The secure content delegation mechanism defined in [SCD] is used to create a separate resource that contains encrypted and integrity protected content. To enable caching, the primary and secondary servers can be the same server.

A client that signals a willingness to support delegation is provided with a response that uses a proxy-enabled out-of-band encoding that behaves identically to the out-of-band encoding defined in [I-D.reschke-http-oob-encoding]. The out-of-band encoding identifies a secondary resource and implicitly indicates that the client is willing to use a proxy and that the server allows this use. The client is then able to request the secondary resource from a proxy cache rather than directly to the origin server.

In this document, the origin server is able to act in the role of the secondary server in [SCD]. However, all of the considerations that apply to having a secondary server host content apply instead to the proxy cache. Thus, integrity and confidentiality protections against the proxy cache are the primary consideration.

2.1. Signaling Presence of a Proxy

Without a clear signal from the client that a caching proxy is present, an origin server is unable to send a response with out-of-band encoding. A value of "out-of-band" in the Accept-Encoding header field only indicates willingness to use the secure content delegation mechanism.

A new "oobp" content encoding is defined. The "oobp" content encoding is identical to the "out-of-band" content encoding, with the following additional conditions:

- A client MUST NOT signal support for "oobp" content encoding unless it is using a proxy cache and it is willing to direct requests to that proxy.

- A server MUST NOT encode a response using the "oobp" content encoding unless it permits the request to be made to a proxy cache.

- The "oobp" content encoding MUST NOT be used to encode the contents of a request. The "out-of-band" content encoding is sufficient for that purpose.

Using a different content encoding name means that a resource using secure content delegation to a secondary server [SCD] does not inadvertently trigger a request via a proxy.

The security properties of delegation via a secondary server and via a caching proxy are similar only to the extent that a third party is involved. However, it might be the case that a secondary server has a stronger relationship with the primary server and additional constraints on its actions, such as contractual limitations. Such constraints might make it feasible to delegate to a secondary server selected by the primary server. A caching proxy might not be considered acceptable in the same way.

The "oobp" content encoding clearly indicates that the client is permitted to retrieve content from a proxy-cache.

Servers that use the "oobp" content encoding MUST include header fields for message integrity and encryption, such as the M-I header field [I-D.thomson-http-mice] or the Crypto-Key header field [I-D.ietf-httpbis-encryption-encoding]. Clients MUST NOT send a request via a proxy if these headers are not present. Absence of these header fields indicate an error on the part of the origin server, since integrity and confidentiality protection are mandatory.

2.2. Proxy Identification and Authentication

This mechanism does not work with a transparent caching proxy. Since the request is made over end-to-end HTTPS in the absence of a proxy, the feature will not be used unless the proxy is known to the client.

A proxy cache MUST therefore be expressly configured or discovered. This produces a name and possibly a port number for the proxy. The proxy MUST be contacted using HTTPS [RFC2818] and authenticated using the configured or discovered domain name.

Issue: What signal do we need from the proxy cache that it supports receiving requests with an "https://" scheme? Can we expect that a proxy cache will happily accept a request for an HTTPS URL? What if they ignore the scheme and send the request in the clear?

3. Performance Optimizations

As noted in [SCD], the secondary request required by out-of-band content encoding imposes a performance penalty. This can be mitigated by priming clients with information about the location and disposition of resources prior to the client making a request. A resource map described in [SCD] might be provided to clients to eliminate the latency involved in making requests of the origin server for resources that might be cached.

3.1. Proxy Cache Priming

A client that makes a request of an origin server via an unprimed proxy cache will suffer additional latency as a consequence of the cache having to make a request to the origin server.

The following options are possible:

- Clients can speculatively make requests to a proxy cache based on information it learns from a resource map, or from hints like the "prefetch" link relation [HINTS]. To avoid a potential waste of resources as a result of receiving complete responses, speculative requests might be limited to HEAD requests; alternatively, HTTP/2 [RFC7540] flow control might be used to allow only limited information to be sent.

- The origin server might provide the proxy cache with "prefetch" link relations in responses to requests for secondary resources. These link relations might identify other resources that the proxy might retrieve speculatively. This does not improve the latency of the initial request, but could improve subsequent requests.

4. Security Considerations

All the considerations of [SCD] apply. In particular, content that is distributed with the assistance of a proxy cache MUST include integrity and confidentiality protection. That means that the M-I header field [I-D.thomson-http-mice] and the Crypto-Key header field [I-D.ietf-httpbis-encryption-encoding] or equivalent information MUST be present in responses that include an out-of-band content encoding.

Clients that receive a response without the information necessary to ensure integrity and confidentiality protection against a proxy cache MUST NOT make a request to a proxy to retrieve that response. Clients could treat such a response as failed. Clients MAY then make the request directly to the origin server, or - if request can be safely retried - retry a request without the out-of-band token in the Accept-Encoding header field.

5. IANA Considerations

This document has no IANA actions. It should.

6. References

6.1. Normative References

[I-D.ietf-httpbis-encryption-encoding]
Thomson, M., "Encrypted Content-Encoding for HTTP", draft-ietf-httpbis-encryption-encoding-02 (work in progress), June 2016.

[I-D.thomson-http-mice]
Thomson, M., "Merkle Integrity Content Encoding", draft-thomson-http-mice-01 (work in progress), June 2016.

[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>.

[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>.

[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>.

[SCD] Ericsson, G., Holmberg, C., and M. Thomson, "An Architecture for Secure Content Delegation using HTTP", February 2016.

6.2. Informative References

[HINTS] Grigorik, I., "Resource Hints", W3C TR , May 2015.

[I-D.reschke-http-oob-encoding]
Reschke, J. and S. Loreto, "'Out-Of-Band' Content Coding for HTTP", draft-reschke-http-oob-encoding-07 (work in progress), July 2016.

[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

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

Goeran AP Eriksson
Ericsson

Email: goran.ap.eriksson@ericsson.com

Christer Holmberg
Ericsson

Email: christer.holmberg@ericsson.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 15, 2019

M. Thomson
Mozilla
J. Yasskin
Google

August 14, 2018

Merkle Integrity Content Encoding
draft-thomson-http-mice-03

Abstract

This memo introduces a content-coding for HTTP that provides progressive integrity for message contents. This integrity protection can be evaluated on a partial representation, allowing a recipient to process a message as it is delivered while retaining strong integrity protection.

Note to Readers

RFC EDITOR: please remove this section before publication

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

The source code and issues list for this draft can be found at https://github.com/martinthomson/http-mice [2].

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 February 15, 2019.

Table of Contents

1. Introduction

Integrity protection for HTTP content is highly valuable. HTTPS [RFC2818] is the most common form of integrity protection deployed, but that requires a direct TLS [RFC8446] connection to a host. However, additional integrity protection might be desirable for some use cases. This might be for additional protection against failures

or attack (see [SRI]) or because content needs to remain unmodified throughout multiple HTTPS-protected exchanges.

This document describes a "mi-sha256" content-encoding (see Section 2) that is a progressive, hash-based integrity check based on Merkle Hash Trees [MERKLE].

The means of conveying the root integrity proof used by this content encoding will depend on deployment requirements. This document defines a digest algorithm (see Section 3) that can carry an integrity proof.

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 "mi-sha256" HTTP Content Encoding

A Merkle Hash Tree [MERKLE] is a structured integrity mechanism that collates multiple integrity checks into a tree. The leaf nodes of the tree contain data (or hashes of data) and non-leaf nodes contain hashes of the nodes below them.

A balanced Merkle Hash Tree is used to efficiently prove membership in large sets (such as in [RFC6962]). However, in this case, a right-skewed tree is used to provide a progressive integrity proof. This integrity proof is used to establish that a given record is part of a message.

The hash function used for "mi-sha256" content encoding is SHA-256 [FIPS180-4]. The integrity proof for all records other than the last is the hash of the concatenation of the record, the integrity proof of all subsequent records, and a single octet with a value of 0x1:

```
proof(r[i]) = SHA-256(r[i] || proof(r[i+1]) || 0x1)
```

The integrity proof for the final record is the hash of the record with a single octet with a value 0x0 appended:

```
proof(r[last]) = SHA-256(r[last] || 0x0)
```

Figure 1 shows the structure of the integrity proofs for a message that is split into 4 blocks: A, B, C, D). As shown, the integrity proof for the entire message (that is, "proof(A)") is derived from the content of the first block (A), plus the value of the proof for the second and subsequent blocks.

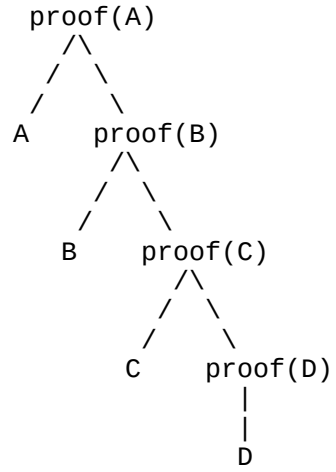


Figure 1: Proof structure for a message with 4 blocks

The final encoded message is formed from the record size and first record, followed by an arbitrary number of tuples of the integrity proof of the next record and then the record itself. Thus, in Figure 1, the body is:

rs || A || proof(B) || B || proof(C) || C || proof(D) || D

Note: The "||" operator is used to represent concatenation.

A message that has a content length less than or equal to the content size does not include any inline proofs. The proof for a message with a single record is simply the hash of the body plus a trailing zero octet.

As a special case, the encoding of an empty payload is itself an empty message (i.e. it omits the initial record size), and its integrity proof is SHA-256("\0").

RFC EDITOR: Please remove the next paragraph before publication.

Implementations of drafts of this specification MUST implement a content encoding named "mi-sha256-##" instead of the "mi-sha256" content encoding specified by the final RFC, with "##" replaced by the draft number being implemented. For example, implementations of draft-thomson-http-mice-03 would implement "mi-sha256-03".

2.1. Content Encoding Structure

In order to produce the final content encoding the content of the message is split into equal-sized records. The final record can contain less than the defined record size.

For non-empty payloads, the record size is included in the first 8 octets of the message as an unsigned 64-bit integer. This refers to the length of each data block.

The final encoded stream comprises of the record size ("rs"), plus a sequence of records, each "rs" octets in length. Each record, other than the last, is followed by a 32 octet proof for the record that follows. This allows a receiver to validate and act upon each record after receiving the proof that precedes it. The final record is not followed by a proof.

Note: This content encoding increases the size of a message by 8 plus 32 octets times the length of the message divided by the record size, rounded up, less one. That is, 8 + 32 * (ceil(length / rs) - 1).

Constructing a message with the "mi-sha256" content encoding requires processing of the records in reverse order, inserting the proof derived from each record before that record.

This structure permits the use of range requests [RFC7233]. However, to validate a given record, a contiguous sequence of records back to the start of the message is needed.

2.2. Validating Integrity Proofs

A receiver of a message with the "mi-sha256" content-encoding applied first attempts to acquire the integrity proof for the first record, "top-proof". If the Digest header field is present with the mi-sha256 parameter, a value might be included there.

The receiver attempts to read the first 8 octets as an unsigned 64-bit integer, "rs". If 8 octets aren’t available then:

- If 0 octets are available, and "top-proof" is SHA-256("\0") (whose base64 encoding is "bjQLnP+zepicpUTmu3gKLHiQHT+zNzh2hRGjBhevoB0="), then return a 0-length decoded payload.

- Otherwise, validation fails.

The remainder of the message is read into records of size "rs" plus 32 octets. The last record is between 1 and "rs" octets in length, if not then validation fails. For each record:

1. Hash the record using SHA-256 with a single octet appended:

   a. All records other than the last have an octet with a value of 0x1 appended.

   b. The last record has an octet with a value of 0x0 appended.

2. Compare the hash with the expected value:

   a. For the first record, the expected value is "top-proof".

   b. For records after the first, the expected value is the last 32 octets of the previous record.

3. If the hash is different, then this record and all subsequent records do not have integrity protection and this process ends.

4. If a record is valid, up to "rs" octets is passed on for processing. In other words, the trailing 32 octets is removed from every record other than the last before being used.

If an integrity check fails, the message SHOULD be discarded and the exchange treated as an error unless explicitly configured otherwise. For clients, treat this as equivalent to a server error; servers SHOULD generate a 400 or other 4xx status code. However, if the integrity proof for the first record is not known, this check SHOULD NOT fail unless explicitly configured to do so.

3. The "mi-sha256" Digest Algorithm

[RFC3230] describes digests applying to "the entire instance associated with the message". The instance corresponds to the "representation" in Section 3 of [RFC7231], but unlike the existing digest algorithms, the "mi-sha256" digest algorithm specifies the top-level digest at the point when the "mi-sha256" content coding (Section 2) is applied or removed from the representation.

When the "mi-sha256" digest algorithm is specified for a representation, the recipient MUST use the base64-decoding (Section 4 of [RFC4648]) of the "mi-sha256" digest as the "top-proof" for the "mi-sha256" content encoding (Section 2.2).

The recipient MUST behave as described by Section 4.2.9 of [I-D.ietf-httpbis-header-structure] if it encounters improper

padding, non-zero padding bits, or non-alphabet characters, where rejecting the data means to reject the representation.

If different mechanisms specify different "top-proof" values for the "mi-sha256" content encoding, the recipient MUST reject the representation.

If "mi-sha256" content coding has not been applied to the representation exactly once (Section 3.1.2.2 of [RFC7231]), the recipient MUST reject the representation.

When rejecting the representation, clients SHOULD treat this as equivalent to a server error, and servers SHOULD generate a 400 or other 4xx status code.

RFC EDITOR: Please remove the next paragraph before publication.

Implementations of drafts of this specification MUST use a digest algorithm named the same as the "mi-sha256-##" content encoding they implement, with the meaning described for "mi-sha256" above.

4. Examples

4.1. Simple Example

The following example contains a short message. This contains just a single record, so there are no inline integrity proofs, just a single value in the mi-sha256 parameter of a Digest header field. The record size is prepended to the message body (shown here in angle brackets).

```
HTTP/1.1 200 OK
Digest: mi-sha256=dcRDgR2GM35DluAV13PzgnG6+pvQwPywfFvAu1UeFrs=
Content-Encoding: mi-sha256
Content-Length: 49

<0x0000000000000029>When I grow up, I want to be a watermelon
```

4.2. Example with Multiple Records

This example shows the same message as above, but with a smaller record size (16 octets). This results in two integrity proofs being included in the representation.

```
PUT /test HTTP/1.1
Host: example.com
Digest: mi-sha256=IVa9shfs0nyKEhHqtB3WVNANJ2Njm5KjQLjRtnbkYJ4=
Content-Encoding: mi-sha256
Content-Length: 113

<0x0000000000000010>When I grow up,
OElbplJlPK+Rv6JNK6p5/515IaoPoZo+2elWL7OQ60A=
I want to be a w
iPMpmgExHPrbEX3/RvwP4d16fWlK4l++p75PUu_KyN0=
atermelon
```

Since the inline integrity proofs contain non-printing characters, these are shown here using the base64 encoding [RFC4648] with new lines between the original text and integrity proofs. Note that there is a single trailing space (0x20) on the first line.

5. Security Considerations

The integrity of an entire message body depends on the means by which the integrity proof for the first record is protected. If this value comes from the same place as the message, then this provides only limited protection against transport-level errors (something that TLS provides adequate protection against).

Separate protection for header fields might be provided by other means if the first record retrieved is the first record in the message, but range requests do not allow for this option.

5.1. Message Truncation

This integrity scheme permits the detection of truncated messages. However, it enables and even encourages processing of messages prior to receiving an complete message. Actions taken on a partial message can produce incorrect results. For example, a message could say "I need some 2mm copper cable, please send 100mm for evaluation purposes" then be truncated to "I need some 2mm copper cable, please send 100m". A network-based attacker might be able to force this sort of truncation by delaying packets that contain the remainder of the message.

Whether it is safe to act on partial messages will depend on the nature of the message and the processing that is performed.

5.2. Algorithm Agility

A new content encoding type is needed in order to define the use of a hash function other than SHA-256.

6. IANA Considerations

6.1. The "mi-sha256" HTTP Content Encoding

This memo registers the "mi-sha256" HTTP content-coding in the HTTP Content Codings Registry, as detailed in Section 2.

- Name: mi-sha256
- Description: A Merkle Hash Tree based content encoding that provides progressive integrity.
- Reference: this specification

6.2. The "mi-sha256" Digest Algorithm

This memo registers the "mi-sha256" digest algorithm in the HTTP Digest Algorithm Values [3] registry:

- Digest Algorithm: mi-sha256
- Description: As specified in Section 3.

7. References

7.1. Normative References

[FIPS180-4] Department of Commerce, National., "NIST FIPS 180-4, Secure Hash Standard", March 2012, <http://csrc.nist.gov/publications/fips/fips180-4/fips-180-4.pdf>.

[I-D.ietf-httpbis-header-structure] Nottingham, M. and P. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-07 (work in progress), July 2018.

[MERKLE] Merkle, R., "A Digital Signature Based on a Conventional Encryption Function", International Crytology Conference - CRYPTO , 1987.

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

[RFC3230] Mogul, J. and A. Van Hoff, "Instance Digests in HTTP", RFC 3230, DOI 10.17487/RFC3230, January 2002, <https://www.rfc-editor.org/info/rfc3230>.

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

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

7.2. Informative References

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

[RFC6962] Laurie, B., Langley, A., and E. Kasper, "Certificate Transparency", RFC 6962, DOI 10.17487/RFC6962, June 2013, <https://www.rfc-editor.org/info/rfc6962>.

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

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

[SRI] Akhawe, D., Braun, F., Marier, F., and J. Weinberger, "Subresource Integrity", W3C CR , November 2015, <https://w3c.github.io/webappsec-subresource-integrity/>.

7.3. URIs

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

[2] https://github.com/martinthomson/http-mice

[3] https://www.iana.org/assignments/http-dig-alg/http-dig-alg.xhtml

Appendix A. Acknowledgements

David Benjamin and Erik Nygren both separately suggested that something like this might be valuable. James Manger and Eric Rescorla provided useful feedback.

Appendix B. FAQ

1. Why not include the first proof in the encoding?

   The requirements for the integrity proof for the first record require a great deal more flexibility than this allows for. Transferring the proof separately is sometimes necessary. Separating the value out allows for that to happen more easily.

2. Why do messages have to be processed in reverse to construct them?

   The final integrity value, no matter how it is derived, has to depend on every bit of the message. That means that there are three choices: both sender and receiver have to process the whole message, the sender has to work backwards, or the receiver has to work backwards. The current form is the best option of the three. The expectation is that this will be useful for content that is generated once and sent multiple times, since the onerous backwards processing requirement can be amortized.

3. Why not just generate a table of hashes?

   An alternative design includes a header that comprises hashes of every block of the message. The final proof is a hash of that table. This has the advantage that the table can be built in any order. The disadvantage is that a receiver needs to store the table while processing content, whereas a chained hash can be processed with a single stored hash worth of state no matter how many blocks are present. The chained hash is also smaller by 32 octets.

Authors' Addresses

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

Jeffrey Yasskin
Google

Email: jyasskin@chromium.org

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

M. Thomson
Mozilla
G. Eriksson
C. Holmberg
Ericsson

October 30, 2016

An Architecture for Secure Content Delegation using HTTP
draft-thomson-http-scd-02

Abstract

An architecture is described for content distribution using a secondary server that might be operated with reduced privileges. This architecture allows a primary server to delegate the responsibility for delivery of the payload of an HTTP response to a secondary server. The secondary server is unable to modify this content. The content is encrypted, which in some cases will prevent the secondary server from learning about the content.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 3, 2017.

Table of Contents

1. Content Distribution Security

The distribution of content on the web at scale is necessarily highly distributed. Large amounts of content needs large numbers of servers. And distributing those servers closer to clients has a significant, positive impact on performance.

A major drawback of existing solutions for content distribution is that a primary server is required to cede control of resources to the secondary server. The secondary server is able to see and modify content that they distribute.

There are few technical mechanisms in place to limit the capabilities of servers that provide content for a given origin. Mechanisms like content security policy [CSP] and sub-resource integrity [SRI] can be used to prevent modification of resources in some contexts, but these mechanisms are limited in what they can protect and they can impose certain operational costs. For the most part, server operators are forced to limit the content that is served on servers that are not

directly under their control or rely on non-technical measures such as contracts and courts to proscribe bad behavior.

1.1. Secure Content Delegation

This document describes how an primary origin server might securely delegate the responsibility for serving content to a secondary server.

The solution comprises three basic components:

- A delegation component allows a primary server to delegate specific resources to another server.
- Integrity attributes ensure that the content cannot be modified by the secondary server.
- Confidentiality protection limits the ability of the secondary server to learn what the content holds.

Note that the guarantees provided by confidentiality protection are not strong, see Section 4 for details.

In addition to these basic components, a fourth mechanism provides a client with the ability to learn resource metadata from the primary server prior to making a request for specific resources. This can dramatically improve performance where a client needs to acquire multiple delegated resources.

No new mechanisms are described in this document; the application of several existing and separately-proposed protocol mechanisms to this problem is described. A primary server can use these mechanisms to take advantage of secondary servers where concerns about security might have otherwise prevented their use. This might be for content that was previously considered too sensitive for third-party distribution, or to access secondary servers that were previously consider insufficiently trustworthy.

1.2. Notational Conventions

The words "MUST", "MUST NOT", "SHOULD", and "MAY" are used in this document. It’s not shouting; when they are capitalized, they have the special meaning defined in [RFC2119].

This document uses the terms client, primary server and secondary server. These terms refer to the three roles played in this architecture. Note that "primary server" as used in this document

encompasses the notion of both an origin server and a gateway as defined in [RFC7230].

2. Out-of-Band Content Encoding

The out-of-band content encoding [I-D.reschke-http-oob-encoding] provides the basis for delegation of content distribution. A request is made to the primary server, but in place of the complete response only response header fields and an out-of-band content encoding is provided. The out-of-band content encoding directs the client to retrieve content from another resource.

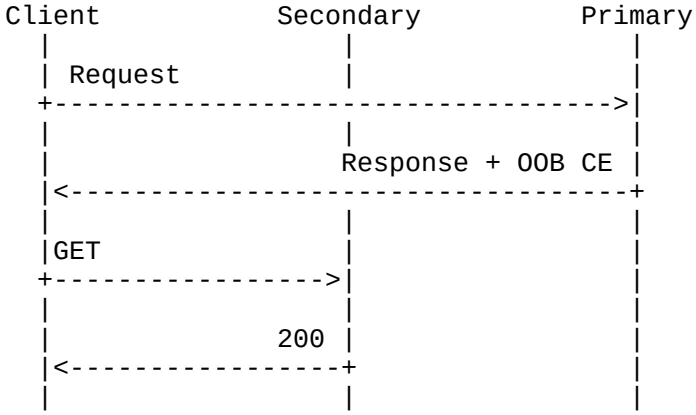


Figure 1: Using Out-of-Band Content Encoding

Out-of-band content encoding behaves much like a redirect. In fact, a redirect was considered as part of the early design, but rejected because without defining a new set of 3xx status codes it would change the effective origin [RFC6454] of the resource. Furthermore, the content encoding specifically preserves header fields sent by the primary server, rejecting any unauthenticated header fields that might be provided by the secondary server.

2.1. Performance Trade-Off

An additional request is necessary to retrieve content. This has a negative impact on latency. However, if the secondary server is positioned close to the client, there are several potential benefits:

Fewer bit-miles: Content hosted in the secondary server that is nearby can be served to those clients without having to traverse a long network path.

Better server resource allocation: Using a dedicated secondary server reduces the load on the primary server, allowing it more capacity for serving other requests.

Better throughput: If a secondary server is closer to a client, more bandwidth might be available for delivery of content when compared with the link between client and primary server.

Lower time to last byte: For some resources, increased bandwidth can counteract the added latency cost of the extra requests, and potentially reduce the time needed to retrieve the entire resource.

The problems of providing integrity protection for content delivered in this fashion is discussed in Section 3; confidentiality protection and its limitations is described in Section 4; and reducing the latency impact of making multiple requests for each resource is described in Section 5.

2.2. Confidentiality of Resource Identity

The URL used to acquire a resource from a secondary server can be unrelated to the URL of the resource that refers to its contents. This allows a primary server to hide the relationship between content in a secondary server and the original resources that is use that content.

Any entity SHOULD be unable to determine the URL of the original resource based on the URL of the secondary server resource alone. This can be achieved by having randomized URLs for secondary resources and maintaining a mapping table, or by using a fixed mapping function with a secret input such as HMAC [RFC2104].

Without other information, this would prevent the secondary server from learning which resources are requested from the primary server by observing the requests that it serves for out-of-band content. While in some cases, information about the resource is obtainable by the secondary server cache, see Section 4, an unpredictable mapping ensures that other protection mechanisms can be effective if possible.

3. Content Integrity

Ensuring that content is not modified by the secondary server is critical. Information that is acquired from the secondary server is not integrity protected and therefore MUST NOT be used without being authenticated.

A cryptographic hash over the content sent in the initial response could be compared against a hash of the content delivered by the secondary server. This is an expansion of the the basic design of [SRI].

A progressive integrity mechanism like the one described in [I-D.thomson-http-mice] ensures that there are no significant performance penalties imposed by the integrity protection. Progressive integrity allows for consumption of content as it is delivered without losing integrity protection.

A response from the primary server could include an M-I header field with an integrity proof, allowing the content to be delivered out-of-band without any additional header fields.

4. Content Confidentiality

Confidentiality protection for content is provided by applying an encryption content encoding [I-D.ietf-httpbis-encryption-encoding] to content before that content is provided to a secondary server.

Much of the value provided by a secondary server derives from its ability to deliver the same content to multiple nearby clients. The more clients that can be delivered the same resource, the greater the efficiency gains. As a result, resources that are provided to many or all clients are the ones that benefit most from caching.

This means that unless a resource has access control mechanisms that would prevent the secondary from accessing a resource, the confidentiality protections provided by encrypting content is limited. A secondary server need only independently request resources from the primary server in order to learn everything about the content it is serving, including the mapping of primary URLs to secondary URLs. For instance, employing a web crawler on a web site might reveal the identity of numerous resources and the location of the any out-of-band content for those resources.

Confidentiality protection allows resources that are protected by client authentication to remain confidential. Confidentiality protection also improves protections against cross-origin theft of confidential data (see Section 7.2).

5. Resource Map

Learning about header fields and out-of-band cache locations for resources in advance of needing to make requests to those resources allows a client to avoid making requests to the primary server. This can greatly improve the performance of applications that make

multiple requests of the same server, such as web browsing or video streaming.

Without defining any new additional protocol mechanisms, HTTP/2 server push [RFC7540] can be used to provide requests, responses and the out-of-band content encoding information describing resources. Since no actual content is included, this requires relatively little data to describe a number of resources. Once this information is available, the client no longer needs to contact the origin server to acquire the described resources.

This approach has some significant deployment drawbacks, so explicit data formats for carrying this data might be defined.

Note: We need a separate draft on these alternative methods.

6. Error Handling

Error handling for clients is described in [I-D.reschke-http-oob-encoding].

For idempotent requests, a second request might be made to the primary server. This request would omit any indication of support for out-of-band content coding from the Accept-Encoding header field, plus a link relation indicating the secondary resource and the reason for failure.

A primary server can use this information to make informed choices about whether to use content delegation.

Non-idempotent requests cannot be safely retried. Therefore, clients cannot retry a a request and provide information about errors to the primary server. For this reason, primary servers SHOULD NOT delegate content for non-idempotent methods.

7. Security Considerations

This document describes a framework whereby content might be distributed to a secondary server, without losing integrity with respect to the content that is distributed.

This design relies on integrity and confidentiality for the request and response made to the primary server. These requests MUST be made using HTTP over TLS (HTTPS) [RFC2818] only. Though there is a lesser requirement for confidentiality, requests made to the secondary server MUST also be secured using HTTPS.

7.1. Confidentiality Protection Limitations

Content that requires only integrity protection can be safely distributed by a third-party using this design. Entities that make a decision about confidentiality for others have often been shown to be incorrect in the past. An incorrect conclusion have serious consequences. Thus the choice of whether confidentiality protection is needed is quite important.

Some confidentiality protection against the secondary server is provided, but that is limited to content that is not otherwise accessible to that server (see Section 4). Only content that has access controls on the primary server that prevent access by the secondary server can retain confidentiality protection.

Content with different access control policies MUST use different keying material for encryption. This prevents a client with access to one resource from acquiring keys that can be used for resources they are not authorized to access.

Clients that wish to retain control over the confidentiality of responses can omit the out-of-band label from the Accept-Encoding header field on requests, thereby indicating that a direct response is necessary.

7.2. Cross-Origin Access

The content delegation creates the possibility that a primary server could adopt remotely hosted content. On the web, this is normally limited by Cross-Origin Resource Sharing [CORS], which requires that a client first request permission to make a resource accessible to another origin.

This document describes a method whereby content hosted on a remote secondary server can be made accessible to another origin. The content of the out-of-band resource is written into the content of a response from the origin. All an origin needs to make this happen is knowledge of the identity of the out-of-band resource, something that might be difficult based on the guidance in Section 2.2, but not infeasible. A client requests this content using any ambient authority available to it (such as HTTP authentication header fields and cookies).

The simplest option for reducing the ability to steal content in this fashion is to require that the origin demonstrate that it knows the content of the resource. Unfortunately, this demonstration is difficult without imposing significant performance penalties, so we

require a lesser assurance: that the origin knows how to decrypt the content.

This makes content confidentiality (Section 4) mandatory and limits the resources that can be stolen by an origin to those that are already encrypted. Most importantly, only resources for which the origin knows the encryption key can be stolen.

For this protection to be effective, origins MUST use different encryption keys for resources with different sets of authorized recipients. Otherwise, an attacker might learn the encryption key for one resource then use that to decrypt a resource that it is not authorized to read.

Resources that rely on signature-based integrity protection are made only marginally more difficult to steal, since the origin needs to learn the signing public key. However, this is not expected to be difficult, since confidentiality protection for public keys. Resources that rely on hash-based integrity protection require that the origin learn the hash of the resource.

7.3. Traffic Analysis

Using a secondary server reveals a great deal of information to the secondary server about resources even if confidentiality protection is effective. The size of responses and the pattern of requests for resources can reveal information about their contents. When used carefully, padding as described in [I-D.ietf-httpbis-encryption-encoding] can obscure the length of responses and reduce the information that the secondary server is able to learn.

A random or unpredictable mapping from the primary resource URL on the primary server to the URL of the content is necessary, see Section 2.2.

Length hiding for header fields on responses from the primary server might be more important when an out-of-band encoding is used, since the body of the response becomes less variable.

Making requests for content to multiple different servers can improve the amount of content length information available to network observers. HTTP/2 multiplexing might have otherwise reduced the exposure of length information, but using out-of-band content encoding could expose lengths for those resources that can be distributed by a secondary server. Note that this is not fundamentally worse than HTTP/1.1 in the absence of pipelining.

Padding in HTTP/2 or encrypted content encoding can be used to further obscure lengths.

8. IANA Considerations

This document has no IANA actions.

9. References

9.1. Normative References

[I-D.reschke-http-oob-encoding]
Reschke, J. and S. Loreto, "'Out-Of-Band' Content Coding for HTTP", draft-reschke-http-oob-encoding-07 (work in progress), July 2016.

[I-D.thomson-http-mice]
Thomson, M., "Merkle Integrity Content Encoding", draft-thomson-http-mice-01 (work in progress), June 2016.

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

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

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

9.2. Informative References

[CORS] van Kesteren, A., "Cross-Origin Resource Sharing", January 2014, <https://www.w3.org/TR/cors/>.

[CSP] West, M., Barth, A., and D. Veditz, "Content Security Policy Level 2", August 2015, <https://w3c.github.io/webappsec-csp/2/>.

[I-D.ietf-httpbis-encryption-encoding]
Thomson, M., "Encrypted Content-Encoding for HTTP", draft-ietf-httpbis-encryption-encoding-02 (work in progress), June 2016.

[I-D.thomson-http-content-signature]
Thomson, M., "Content-Signature Header Field for HTTP", draft-thomson-http-content-signature-00 (work in progress), July 2015.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

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

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

[SRI] Akhawe, D., Braun, F., Marier, F., and J. Weinberger, "Subresource Integrity", November 2015, <https://w3c.github.io/webappsec-subresource-integrity>.

Appendix A. Acknowledgements

Magnus Westerlund noted the potential for a violation of the cross origin protections offered in browsers.

Authors' Addresses

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

Goeran AP Eriksson
Ericsson

Email: goran.ap.eriksson@ericsson.com

Christer Holmberg
Ericsson

Email: christer.holmberg@ericsson.com

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

M. Thomson
Mozilla
M. Bishop
Microsoft

March 14, 2016

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

Abstract

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

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

Status of This Memo

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

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

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

This Internet-Draft will expire on September 15, 2016.

Table of Contents

1. Introduction

Many existing HTTP [RFC7230] servers have different authentication requirements for the different resources they serve. Of the bountiful authentication options available for authenticating HTTP requests, client certificates present a unique challenge for resource-specific authentication requirements because of the interaction with the underlying TLS RFC5246 [I-D.ietf-tls-tls13] layer.

For servers that wish to use client certificates to authenticate users, they might request client authentication during or immediately after the TLS handshake. However, if not all users or resources need certificate-based authentication, a request for a certificate has the unfortunate consequence of triggering the client to seek a certificate. Such a request can result in a poor experience, particularly when sent to a client that does not expect the request.

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

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

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

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

1.1. Reactive Certificate Authentication in HTTP/1.1

1.1.1. Using TLS 1.2 and previous

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

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

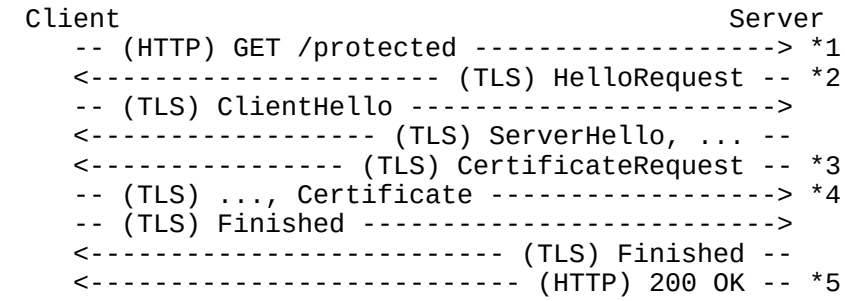


Figure 1: HTTP/1.1 Reactive Certificate Authentication with TLS 1.2

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

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

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

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

1.1.2. Using TLS 1.3

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

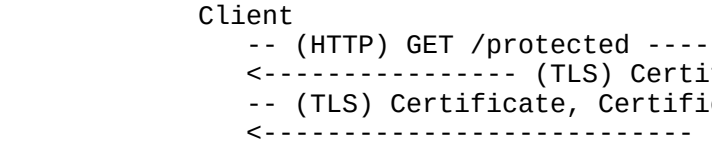


Figure 2: HTTP/1.1 Reactive Certificate Authentication with TLS 1.3

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

1.2. Reactive Client Authentication in HTTP/2

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

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

Thus, the minimum necessary mechanism to support reactive certificate authentication in HTTP/2 is an identifier that can be use to correlate an HTTP request with a request for a certificate.

Such an identifier could be added to TLS 1.2 by means of an extension, but many TLS 1.2 implementations do not permit application data to continue during a renegotiation. This is problematic for a multiplexed protocol like HTTP/2. Instead, this draft proposes bringing the TLS 1.3 CertificateRequest, Certificate, and CertificateVerify messages into HTTP/2 frames, making client certificate authentication TLS-version-agnostic.

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

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

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

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

The exchange then looks like this:

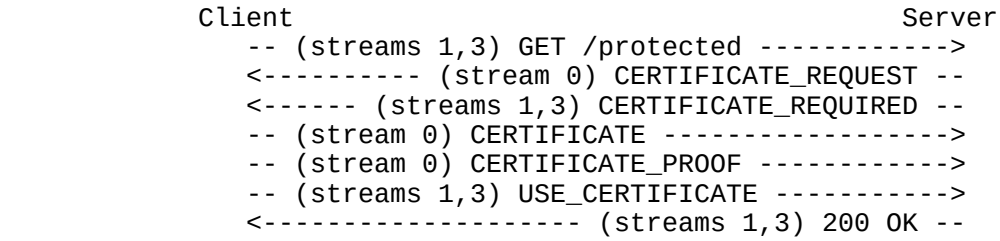


Figure 3: HTTP/2 Reactive Certificate Authentication

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

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

1.3. Terminology

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

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

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

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

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

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

2.1. The CERTIFICATE_REQUIRED frame

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

The "CERTIFICATE_REQUIRED" frame contains 1 octet, which is the authentication request identifier. A client that receives a "CERTIFICATE_REQUIRED" of any other length MUST treat this as a stream error of type "PROTOCOL_ERROR". Frames with identical request identifiers refer to the same "CERTIFICATE_REQUEST".

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

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

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

2.2. The USE_CERTIFICATE Frame

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

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

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

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

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

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

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

2.3. The CERTIFICATE_REQUEST Frame

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

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

The "CERTIFICATE_REQUEST" frame MUST be sent on stream zero. A "CERTIFICATE_REQUEST" frame received on any other stream MUST be rejected with a stream error of type "PROTOCOL_ERROR".

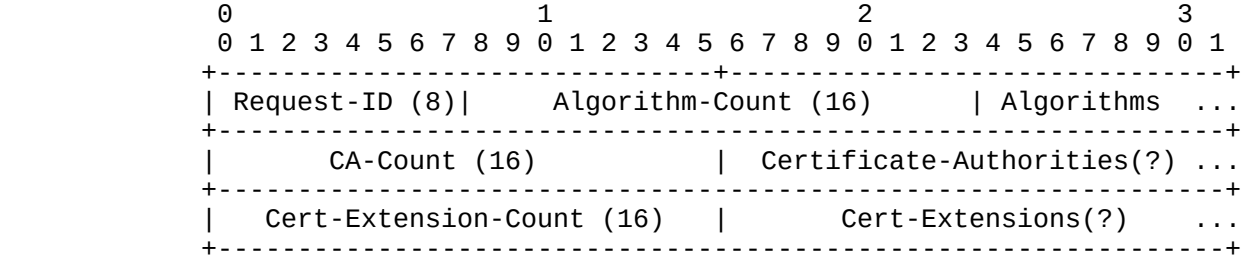


Figure 4: CERTIFICATE_REQUEST frame payload

The frame contains the following fields:

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

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

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

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

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

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

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

2.4. The CERTIFICATE frame

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

The "CERTIFICATE" frame defines no flags.

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

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

The "CERTIFICATE" frame MUST be sent on stream zero. A "CERTIFICATE" frame received on any other stream MUST be rejected with a stream error of type "PROTOCOL_ERROR".

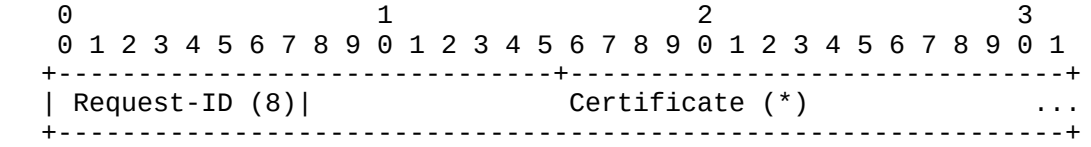


Figure 5: CERTIFICATE frame payload

The fields defined by the "CERTIFICATE" frame are:

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

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

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

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

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

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

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

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

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

2.5. The CERTIFICATE_PROOF Frame

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

The "CERTIFICATE_PROOF" frame defines one flag:

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

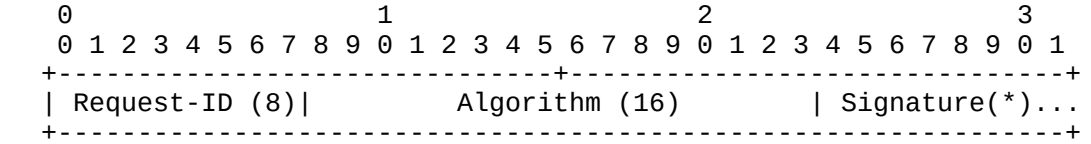


Figure 6: CERTIFICATE_PROOF frame payload

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

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

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

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

  - Length: 64 bytes

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

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

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

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

3. Indicating failures during HTTP-Layer Certificate Authentication

Because this draft permits client certificates to be exchanged at the HTTP framing layer instead of the TLS layer, several certificate-related errors which are defined at the TLS layer might now occur at the HTTP framing layer. In this section, those errors are restated and added to the HTTP/2 error code registry.

BAD_CERTIFICATE (0xERROR-TBD1): A certificate was corrupt, contained signatures that did not verify correctly, etc.

UNSUPPORTED_CERTIFICATE (0xERROR-TBD2): A certificate was of an unsupported type or did not contain required extensions

CERTIFICATE_REVOKED (0xERROR-TBD3): A certificate was revoked by its signer

CERTIFICATE_EXPIRED (0xERROR-TBD4): A certificate has expired or is not currently valid

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

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

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

As described in [RFC7540], implementations MAY choose to treat a stream error as a connection error at any time. Of particular note, a stream error cannot occur on stream 0, which means that implementations cannot send non-session errors in response to "CERTIFICATE_REQUEST" and "CERTIFICATE" frames. Implementations which do not wish to terminate the connection MAY either send relevant errors on any stream which references the failing certificate in question or process the requests as unauthenticated and provide error information at the HTTP semantic layer.

4. Indicating Support for HTTP-Layer Certificate Authentication

Clients that support HTTP-layer certificate authentication indicate this using the HTTP/2 "SETTINGS_HTTP_CERT_AUTH" (0xSETTING-TBD) setting.

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

5. Security Considerations

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

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

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

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

Client implementations need to carefully consider the impact of setting the "AUTOMATIC_USE" flag. This flag is a performance optimization, permitting the client to avoid a round-trip on each request where the server checks for certificate authentication. However, once this flag has been sent, the client has zero knowledge about whether the server will use the referenced cert for any future request, or even for an existing request which has not yet completed. Clients MUST NOT set this flag on any certificate which is not appropriate for currently-in-flight requests, and MUST NOT make any future requests on the same connection which they do not intend to have associated with the provided certificate.

Implementations need to be aware of the potential for confusion about the state of a connection. The presence or absence of a validated client certificate can change during the processing of a request, potentially multiple times, as "USE_CERTIFICATE" frames are received. A server that uses certificate authentication needs to be prepared to reevaluate the authorization state of a request as the set of certificates changes.

6. IANA Considerations

The HTTP/2 "SETTINGS_HTTP_CERT_AUTH" setting is registered in Section 6.1. Five frame types are registered in Section 6.2. Six error codes are registered in Section 6.3.

6.1. HTTP/2 SETTINGS_HTTP_CERT_AUTH Setting

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

Name: SETTINGS_HTTP_CERT_AUTH

Code: 0xSETTING-TBD

Initial Value: 0

Specification: This document.

6.2. New HTTP/2 Frames

Four new frame types are registered in the "HTTP/2 Frame Types" registry established in [RFC7540].

6.2.1. CERTIFICATE_REQUIRED

Frame Type: CERTIFICATE_REQUIRED

Code: 0xFRAME-TBD1

Specification: This document.

6.2.2. CERTIFICATE_REQUEST

Frame Type: CERTIFICATE_REQUEST

Code: 0xFRAME-TBD2

Specification: This document.

6.2.3. CERTIFICATE

Frame Type: CERTIFICATE

Code: 0xFRAME-TBD3

Specification: This document.

6.2.4. CERTIFICATE_PROOF

Frame Type: CERTIFICATE_PROOF

Code: 0xFRAME-TBD4

Specification: This document.

6.2.5. USE_CERTIFICATE

Frame Type: USE_CERTIFICATE

Code: 0xFRAME-TBD5

Specification: This document.

6.3. New HTTP/2 Error Codes

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

6.3.1. BAD_CERTIFICATE

Name: BAD_CERTIFICATE

Code: 0xERROR-TBD1

Specification: This document.

6.3.2. UNSUPPORTED_CERTIFICATE

Name: UNSUPPORTED_CERTIFICATE

Code: 0xERROR-TBD2

Specification: This document.

6.3.3. CERTIFICATE_REVOKED

Name: CERTIFICATE_REVOKED

Code: 0xERROR-TBD3

Specification: This document.

6.3.4. CERTIFICATE_EXPIRED

Name: CERTIFICATE_EXPIRED

Code: 0xERROR-TBD4

Specification: This document.

6.3.5. BAD_SIGNATURE

Name: BAD_SIGNATURE

Code: 0xERROR-TBD5

Specification: This document.

6.3.6. CERTIFICATE_GENERAL

Name: CERTIFICATE_GENERAL

Code: 0xERROR-TBD6

Specification: This document.

7. Acknowledgements

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

8. Normative References

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

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

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

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

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

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

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

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

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

Authors' Addresses

Martin Thomson
Mozilla

Email: martin.thomson@gmail.com

Mike Bishop
Microsoft

Email: michael.bishop@microsoft.com

HTTPbis
Internet-Draft
Updates: 6265 (if approved)
Intended status: Standards Track
Expires: September 4, 2016

E. Wright
Shopify
S. Huang
M. West
Google, Inc

March 3, 2016

A Retention Priority Attribute for HTTP Cookies
draft-west-cookie-priority-00

Abstract

This document defines the "Priority" attribute for HTTP cookies. This attribute allows servers to specify a retention priority for HTTP cookies that will be respected by user agents during cookie eviction.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 4, 2016.

Copyright Notice

Table of Contents

1. Introduction

This document defines the "Priority" attribute for HTTP cookies. Using the "Priority" attribute, servers may indicate that certain cookies should be protected, and others preferentially deleted. When a user agent evicts cookies in the enforcement of a per-domain quota, lower priority cookies will be deleted first, potentially preserving higher-priority cookies that would otherwise have been deleted according to the rules of [RFC6265].

2. Terminology and notation

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

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

Two sequences of octets are said to case-insensitively match each other if and only if they are equivalent under the "i;ascii-casemap" collation defined in [RFC4790].

3. Overview

This section outlines a way for an origin server to indicate the retention priority for individual cookies and for the user agent to respect retention priorities during cookie eviction.

To indicate a cookie’s retention priority, the origin server includes a "Priority" attribute in the "Set-Cookie" HTTP response header. A value of "Low" indicates that the cookie should be given lower retention priority (evicted prior to other cookies). A value of "High" indicates that a cookie should be given higher retention priority (evicted after other cookies). The value of "Medium" corresponds to the default behavior.

In order to prevent starvation of functionality dependent on low- and medium-priority cookies, a fraction of the cookie quota should be reserved for them.

[RFC6265], Section 5.3, describes a strategy that user agents must follow when "removing excess cookies". A user agent implementing the current specification for a retention priority attribute will implement an extended priority order, dividing the second priority ("Cookies that share a domain field with more than a predetermined number of other cookies") into multiple levels corresponding to the three retention priorities.

As a result, when the cookies for a given domain exceed user agent limits, cookies with low priority will be evicted first, followed by medium and high priority cookies.

3.1. Examples

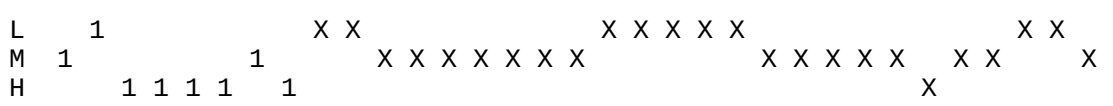
Using the Priority attribute, an origin server may assign a retention priority to a cookie stored by a user agent. For example, the origin server may prioritize the retention of session security tokens while indicating that superficial data such as a user’s favorite color should be discarded in preference to other data.

The following figure illustrates a series of 33 cookies received by a user-agent from the example.com server during one or more HTTP responses. Each cookie is represented by an L, M, or H indicating low, medium, or high priority, respectively.

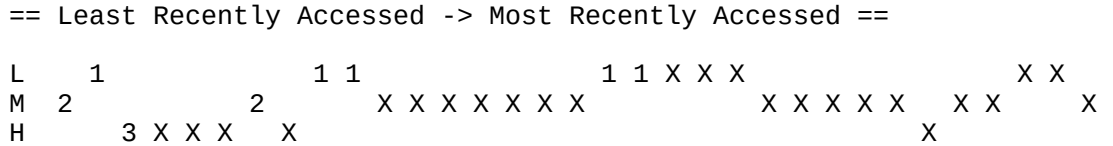
```
== Least Recently Accessed -> Most Recently Accessed ==

   M L H H H H M H L L M M M M M M M L L L L L M M M M M H M M L L M
```

Assume that the user agent is configured to evict all but 25 cookies from a domain when the number of cookies exceeds 31. [RFC6265] specifies the following eviction order. Cookies are separated in the vertical axis by priority. Evicted cookies are labeled with a '1' and retained cookies with an 'X'.



A user agent that implements the current specification, reserving room for 5 low-, 15 medium-, and 5 high-priority cookies, would implement a modified eviction order as follows. '1', '2', and '3' indicate cookies evicted during various phases of the algorithm.



Of note is that the retention priority does not impact the relative eviction priority of cookies being evicted due to the global threshold (i.e., once no domain exceeds the per-domain threshold). Furthermore, this new attribute has no effect on domains that do not send it.

4. Server Requirements

This section describes the syntax and semantics of the "Priority" cookie attribute.

4.1. Syntax

The Set-Cookie HTTP response header syntax is defined in [RFC6265], Section 4.1.1. The grammar defined therein provides for tokens of type 'extension-av'. The Priority attribute is a subset of 'extension-av' that may appear zero or one times in a given 'set-cookie-header'. If it appears, the 'priority' attribute must conform to the following grammar:

```
priority-av    = "Priority=" priority-value
priority-value = "Low" / "Medium" / "High"
```

4.2. Semantics (Non-Normative)

This section describes a simplified semantics of the "Priority" attribute in the "Set-Cookie" HTTP response header. The full semantics are described in Section 5.

4.3. The 'Priority' Attribute

The "Priority" attribute indicates a retention priority relative to other cookies from the same domain as the cookie carrying the attribute. During cookie eviction in enforcement of per-domain cookie limits, "Low" priority cookies will be evicted before "Medium" and "Medium" before "High". Cookies without a specified priority are considered to have "Medium" priority.

5. User Agent Requirements

[RFC6265], Section 5.2 describes how user agents must parse the value of the "Set-Cookie" HTTP response header. This specification provides additional processing steps that user agents must follow when they encounter a "Priority" attribute.

"Set-Cookie" headers that do not specify the "Priority" attribute MUST be treated as if the attribute was present and had the value Medium.

5.1. The 'Priority' Attribute

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

If the "attribute-value" case-insensitively matches the string "Low", the cookie is assigned a low retention priority.

If the "attribute-value" case-insensitively matches the string "Medium", the cookie is assigned a medium retention priority.

If the "attribute-value" case-insensitively matches the string "High", the cookie is assigned a high retention priority.

Otherwise, the cookie is assigned a medium retention priority.

5.2. Storage Model

[RFC6265], Section 6.1 recommends that user agents set limits on the number of cookies they will store. [RFC6265], Section 5.3, describes a strategy that user must follow when "removing excess cookies".

A user agent implementing the current specification for a retention priority attribute will set aside a small portion of its storage quota for low-priority cookies, and another portion for medium-priority cookies. During eviction, compatible user agents will implement an extended priority order, dividing the second priority ("Cookies that share a domain field with more than a predetermined number of other cookies") into three, according to the retention priorities and the space reserved for them. Assuming that no other extensions ammend the rules defined in [RFC6265], a compatible user agent MUST therefore evict cookies in the following priority order (L and M refer to the amount of space reserved for low- and medium-priority cookies respectively, per domain). As per [RFC6265], within each category the least-recently accessed cookies should be deleted first.

1. Expired cookies.

2. Cookies with a low retention priority that share a domain field with more than a predetermined number of other cookies, excluding the first L low-priority cookies from that domain.

3. Cookies with a low or medium retention priority that share a domain field with more than a predetermined number of other cookies, excluding the first L + M low- and medium-priority cookies from that domain.

4. Cookies that share a domain field with more than a predetermined number of other cookies.

5. All cookies.

6. Implementation Considerations

This specification extends [RFC6265] in order to enable improved behaviour when servers are unable or unwilling to keep the number of distinct cookies served by their domains within the limits of user agents. As this specification is unlikely to ever achieve universal adoption by user agents, servers SHOULD gracefully degrade if their specifed cookie retention priorities are not respected.

7. References

7.1. Normative References

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

[RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, DOI 10.17487/RFC4790, March 2007, <http://www.rfc-editor.org/info/rfc4790>.

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

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

7.2. Informative References

[Wright2013] Wright, E. and S. Huang, "A Retention Priority Attribute for HTTP Cookies", n.d., <https://docs.google.com/a/google.com/file/d/0B3o1IlTKoADVRllKWGlyWGxIVTg/edit>.

Appendix A. Acknowledgements

This document is based heavily on an earlier draft written by Erik Wright and Samuel Huang [Wright2013], and the experimentation done in cooperation with Google’s login team.

Authors’ Addresses

Erik Wright
Shopify

Samuel Huang
Google, Inc

Email: huangs@google.com

Mike West
Google, Inc

Email: mkwst@google.com

HTTPbis
Internet-Draft
Updates: 6265 (if approved)
Intended status: Standards Track
Expires: October 8, 2016

M. West
Google, Inc
M. Goodwin
Mozilla

April 6, 2016

Same-site Cookies
draft-west-first-party-cookies-07

Abstract

This document updates RFC6265 by defining a "SameSite" attribute which allows servers to assert that a cookie ought not to be sent along with cross-site requests. This assertion allows user agents to mitigate the risk of cross-origin information leakage, and provides some protection against cross-site request forgery attacks.

Status of This Memo

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

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

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

This Internet-Draft will expire on October 8, 2016.

Table of Contents

1. Introduction

Section 8.2 of [RFC6265] eloquently notes that cookies are a form of ambient authority, attached by default to requests the user agent sends on a user's behalf. Even when an attacker doesn't know the contents of a user's cookies, she can still execute commands on the user's behalf (and with the user's authority) by asking the user agent to send HTTP requests to unwary servers.

Here, we update [RFC6265] with a simple mitigation strategy that allows servers to declare certain cookies as "same-site", meaning they should not be attached to "cross-site" requests (as defined in section 2.1).

Note that the mechanism outlined here is backwards compatible with the existing cookie syntax. Servers may serve these cookies to all user agents; those that do not support the "SameSite" attribute will simply store a cookie which is attached to all relevant requests, just as they do today.

1.1. Goals

These cookies are intended to provide a solid layer of defense-in-depth against attacks which require embedding an authenticated request into an attacker-controlled context:

1. Timing attacks which yield cross-origin information leakage (such as those detailed in [pixel-perfect]) can be substantially mitigated by setting the "SameSite" attribute on authentication cookies. The attacker will only be able to embed unauthenticated resources, as embedding mechanisms such as "<iframe>" will yield cross-site requests.

2. Cross-site script inclusion (XSSI) attacks are likewise mitigated by setting the "SameSite" attribute on authentication cookies. The attacker will not be able to include authenticated resources via "<script>" or "<link>", as these embedding mechanisms will likewise yield cross-site requests.

3. Cross-site request forgery (CSRF) attacks which rely on top-level navigation (HTML "<form>" POSTs, for instance) can also be mitigated by treating these navigational requests as "cross-site".

4. Same-site cookies have some marginal value for policy or regulatory purposes, as cookies which are not delivered with cross-site requests cannot be directly used for tracking purposes. It may be valuable for an origin to assert that its cookies should not be sent along with cross-site requests in order to limit its exposure to non-technical risk.

1.2. Examples

Same-site cookies are set via the "SameSite" attribute in the "Set-Cookie" header field. That is, given a server’s response to a user agent which contains the following header field:

```
Set-Cookie: SID=31d4d96e407aad42; SameSite=Strict
```

Subsequent requests from that user agent can be expected to contain the following header field if and only if both the requested resource and the resource in the top-level browsing context match the cookie.

2. Terminology and notation

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

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

Two sequences of octets are said to case-insensitively match each other if and only if they are equivalent under the "i;ascii-casemap" collation defined in [RFC4790].

The terms "active document", "ancestor browsing context", "browsing context", "document", "WorkerGlobalScope", "sandboxed origin browsing context flag", "parent browsing context", "the worker's Documents", "nested browsing context", and "top-level browsing context" are defined in [HTML].

"Service Workers" are defined in the Service Workers specification [SERVICE-WORKERS].

The term "origin", the mechanism of deriving an origin from a URI, and the "the same" matching algorithm for origins are defined in [RFC6454].

"Safe" HTTP methods include "GET", "HEAD", "OPTIONS", and "TRACE", as defined in Section 4.2.1 of [RFC7231].

The term "public suffix" is defined in a note in Section 5.3 of [RFC6265] as "a domain that is controlled by a public registry". For example, "example.com"'s public suffix is "com". User agents SHOULD use an up-to-date public suffix list, such as the one maintained by Mozilla at [PSL].

An origin's "registrable domain" is the origin's host's public suffix plus the label to its left. That is, "https://www.example.com"'s registrable domain is "example.com". This concept is defined more rigorously in [PSL].

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

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

A request is "same-site" if its target's URI's origin's registrable domain is an exact match for the request's initiator's "site for cookies", and "cross-site" otherwise. To be more precise, for a

given request ("request"), the following algorithm returns "same-site" or "cross-site":

1. If "request"'s client is "null", return "same-site".

2. Let "site" be "request"'s client's "site for cookies" (as defined in the following sections).

3. Let "target" be the registrable domain of "request"'s current url.

4. If "site" is an exact match for "target", return "same-site".

5. Return "cross-site".

2.1.1. Document-based requests

The URI displayed in a user agent's address bar is the only security context directly exposed to users, and therefore the only signal users can reasonably rely upon to determine whether or not they trust a particular website. The registrable domain of that URI's origin represents the context in which a user most likely believes themselves to be interacting. We'll label this domain the "top-level site".

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

For documents which are displayed in nested browsing contexts, we need to audit the origins of each of a document's ancestor browsing contexts' active documents in order to account for the "multiple-nested scenarios" described in Section 4 of [RFC7034]. These document's "site for cookies" is the top-level site if and only if the document and each of its ancestor documents' origins have the same registrable domain as the top-level site. Otherwise its "site for cookies" is the empty string.

Given a Document ("document"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Let "top-document" be the active document in "document"'s browsing context's top-level browsing context.

2. Let "top-origin" be the origin of "top-document"'s URI if "top-document"'s sandboxed origin browsing context flag is set, and "top-document"'s origin otherwise.

3. Let "documents" be a list containing "document" and each of "document"'s ancestor browsing contexts' active documents.

4. For each "item" in "documents":

    1. Let "origin" be the origin of "item"'s URI if "item"'s sandboxed origin browsing context flag is set, and "item"'s origin otherwise.

    2. If "origin"'s host's registrable domain is not an exact match for "top-origin"'s host's registrable domain, return the empty string.

5. Return "top-site".

2.1.2. Worker-based requests

Worker-driven requests aren't as clear-cut as document-driven requests, as there isn't a clear link between a top-level browsing context and a worker. This is especially true for Service Workers [SERVICE-WORKERS], which may execute code in the background, without any document visible at all.

Note: The descriptions below assume that workers must be same-origin with the documents that instantiate them. If this invariant changes, we'll need to take the worker's script's URI into account when determining their status.

2.1.2.1. Dedicated and Shared Workers

Dedicated workers are simple, as each dedicated worker is bound to one and only one document. Requests generated from a dedicated worker (via "importScripts", "XMLHttpRequest", "fetch()", etc) define their "site for cookies" as that document's "site for cookies".

Shared workers may be bound to multiple documents at once. As it is quite possible for those documents to have distinct "site for cookie" values, the worker's "site for cookies" will be the empty string in cases where the values diverge, and the shared value in cases where the values agree.

Given a WorkerGlobalScope ("worker"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Let "site" be "worker"'s origin's host's registrable domain.

2. For each "document" in "worker"'s Documents:

1. Let "document-site" be "document"'s "site for cookies" (as defined in Section 2.1.1).

2. If "document-site" is not an exact match for "site", return the empty string.

3. Return "site".

2.1.2.2. Service Workers

Service Workers are more complicated, as they act as a completely separate execution context with only tangential relationship to the Document which registered them.

Requests which simply pass through a service worker will be handled as described above: the request's client will be the Document or Worker which initiated the request, and its "site for cookies" will be those defined in Section 2.1.1 and Section 2.1.2.1

Requests which are initiated by the Service Worker itself (via a direct call to "fetch()", for instance), on the other hand, will have a client which is a ServiceWorkerGlobalScope. Its "site for cookies" will be the registrable domain of the Service Worker's URI.

Given a ServiceWorkerGlobalScope ("worker"), the following algorithm returns its "site for cookies" (either a registrable domain, or the empty string):

1. Return "worker"'s origin's host's registrable domain.

3. Server Requirements

This section describes extensions to [RFC6265] necessary to implement the server-side requirements of the "SameSite" attribute.

3.1. Grammar

Add "SameSite" to the list of accepted attributes in the "Set-Cookie" header field's value by replacing the "cookie-av" token definition in Section 4.1.1 of [RFC6265] with the following ABNF grammar:

```
cookie-av      = expires-av / max-age-av / domain-av /
                 path-av / secure-av / httponly-av /
                 samesite-av / extension-av
samesite-av    = "SameSite" / "SameSite=" samesite-value
samesite-value = "Strict" / "Lax"
```

3.2. Semantics of the "SameSite" Attribute (Non-Normative)

The "SameSite" attribute limits the scope of the cookie such that it will only be attached to requests if those requests are "same-site", as defined by the algorithm in Section 2.1. For example, requests for "https://example.com/sekrit-image" will attach same-site cookies if and only if initiated from a context whose "site for cookies" is "example.com".

If the "SameSite" attribute's value is "Strict", or if the value is invalid, the cookie will only be sent along with "same-site" requests. If the value is "Lax", the cookie will be sent with "same-site" requests, and with "cross-site" top-level navigations, as described in Section 4.1.1.

The changes to the "Cookie" header field suggested in Section 4.3 provide additional detail.

4. User Agent Requirements

This section describes extensions to [RFC6265] necessary in order to implement the client-side requirements of the "SameSite" attribute.

4.1. The "SameSite" attribute

The following attribute definition should be considered part of the the "Set-Cookie" algorithm as described in Section 5.2 of [RFC6265]:

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

1. If "cookie-av"'s "attribute-value" is not a case-sensitive match for "Strict" or "Lax", ignore the "cookie-av".

2. Let "enforcement" be "Lax" if "cookie-av"'s "attribute-value" is a case-insensitive match for "Lax", and "Strict" otherwise.

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

4.1.1. "Strict" and "Lax" enforcement

By default, same-site cookies will not be sent along with top-level navigations. As discussed in Section 5.2, this might or might not be compatible with existing session management systems. In the interests of providing a drop-in mechanism that mitigates the risk of CSRF attacks, developers may set the "SameSite" attribute in a "Lax"

enforcement mode that carves out an exception which sends same-site cookies along with cross-site requests if and only if they are top-level navigations which use a "safe" (in the [RFC7231] sense) HTTP method.

Lax enforcement provides reasonable defense in depth against CSRF attacks that rely on unsafe HTTP methods (like "POST"), but do not offer a robust defense against CSRF as a general category of attack:

1. Attackers can still pop up new windows or trigger top-level navigations in order to create a "same-site" request (as described in section 2.1), which is only a speedbump along the road to exploitation.

2. Features like "<link rel='prerender'>" [prerendering] can be exploited to create "same-site" requests without the risk of user detection.

When possible, developers should use a session management mechanism such as that described in Section 5.2 to mitigate the risk of CSRF more completely.

4.2. Monkey-patching the Storage Model

Note: There's got to be a better way to specify this. Until I figure out what that is, monkey-patching!

Alter Section 5.3 of [RFC6265] as follows:

1. Add "samesite-flag" to the list of fields stored for each cookie. This field's value is one of "None", "Strict", or "Lax".

2. Before step 11 of the current algorithm, add the following:

    1. If the "cookie-attribute-list" contains an attribute with an "attribute-name" of "SameSite", set the cookie's "samesite-flag" to "attribute-value" ("Strict" or "Lax"). Otherwise, set the cookie's "samesite-flag" to "None".

    2. If the cookie's "samesite-flag" is not "None", and the request which generated the cookie's client's "site for cookies" is not an exact match for "request-uri"'s host's registrable domain, then abort these steps and ignore the newly created cookie entirely.

4.3. Monkey-patching the "Cookie" header

Note: There's got to be a better way to specify this. Until I figure out what that is, monkey-patching!

Alter Section 5.4 of [RFC6265] as follows:

1. Add the following requirement to the list in step 1:

    * If the cookie's "samesite-flag" is not "None", and the HTTP request is cross-site (as defined in Section 2.1 then exclude the cookie unless all of the following statements hold:

        1. "samesite-flag" is "Lax"

        2. The HTTP request's method is "safe".

        3. The HTTP request's target browsing context is a top-level browsing context.

Note that the modifications suggested here concern themselves only with the "site for cookies" of the request's client, and the registrable domain of the resource being requested. The cookie's "domain", "path", and "secure" attributes do not come into play for these comparisons.

5. Authoring Considerations

5.1. Defense in depth

"SameSite" cookies offer a robust defense against CSRF attack when deployed in strict mode, and when supported by the client. It is, however, prudent to ensure that this designation is not the extent of a site's defense against CSRF, as same-site navigations and submissions can certainly be executed in conjunction with other attack vectors such as cross-site scripting.

Developers are strongly encouraged to deploy the usual server-side defenses (CSRF tokens, ensuring that "safe" HTTP methods are idempotent, etc) to mitigate the risk more fully.

Additionally, client-side techniques such as those described in [app-isolation] may also prove effective against CSRF, and are certainly worth exploring in combination with "SameSite" cookies.

5.2. Top-level Navigations

Setting the "SameSite" attribute in "strict" mode provides robust defense in depth against CSRF attacks, but has the potential to confuse users unless sites' developers carefully ensure that their session management systems deal reasonably well with top-level navigations.

Consider the scenario in which a user reads their email at MegaCorp Inc's webmail provider "https://example.com/". They might expect that clicking on an emailed link to "https://projects.com/secret/project" would show them the secret project that they're authorized to see, but if "projects.com" has marked their session cookies as "SameSite", then this cross-site navigation won't send them along with the request. "projects.com" will render a 404 error to avoid leaking secret information, and the user will be quite confused.

Developers can avoid this confusion by adopting a session management system that relies on not one, but two cookies: one conceptualy granting "read" access, another granting "write" access. The latter could be marked as "SameSite", and its absence would provide a reauthentication step before executing any non-idempotent action. The former could drop the "SameSite" attribute entirely, or choose the "Lax" version of enforcement, in order to allow users access to data via top-level navigation.

5.3. Mashups and Widgets

The "SameSite" attribute is inappropriate for some important use-cases. In particular, note that content intended for embedding in a cross-site contexts (social networking widgets or commenting services, for instance) will not have access to such cookies. Cross-site cookies may be required in order to provide seamless functionality that relies on a user's state.

Likewise, some forms of Single-Sign-On might require authentication in a cross-site context; these mechanisms will not function as intended with same-site cookies.

6. Privacy Considerations

6.1. Server-controlled

Same-site cookies in and of themselves don't do anything to address the general privacy concerns outlined in Section 7.1 of [RFC6265]. The attribute is set by the server, and serves to mitigate the risk of certain kinds of attacks that the server is worried about. The user is not involved in this decision. Moreover, a number of side-

channels exist which could allow a server to link distinct requests even in the absence of cookies. Connection and/or socket pooling, Token Binding, and Channel ID all offer explicit methods of identification that servers could take advantage of.

6.2. Pervasive Monitoring

As outlined in [RFC7258], pervasive monitoring is an attack. Cookies play a large part in enabling such monitoring, as they are responsible for maintaining state in HTTP connections. We considered restricting same-site cookies to secure contexts [secure-contexts] as a mitigation but decided against doing so, as this feature should result in a strict reduction in the number of cookies floating around in cross-site contexts. That is, even if "http://not-example.com" embeds a resource from "http://example.com/", that resource will not be "same-site", and "http://example.com"'s cookies simply cannot be used to correlate user behavior across distinct origins.

7. References

7.1. Normative References

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

[HTML] Hickson, I., Pieters, S., van Kesteren, A., Jaegenstedt, P., and D. Denicola, "HTML", n.d., <https://html.spec.whatwg.org/>.

[PSL] "Public Suffix List", n.d., <https://publicsuffix.org/list/>.

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

[RFC4790] Newman, C., Duerst, M., and A. Gulbrandsen, "Internet Application Protocol Collation Registry", RFC 4790, DOI 10.17487/RFC4790, March 2007, <http://www.rfc-editor.org/info/rfc4790>.

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

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

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

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

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

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

7.2. Informative References

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

[pixel-perfect] Stone, P., "Pixel Perfect Timing Attacks with HTML5", n.d., <http://www.contextis.com/documents/2/Browser_Timing_Attacks.pdf>.

[prerendering] Bentzel, C., "Chrome Prerendering", n.d., <https://www.chromium.org/developers/design-documents/prerender>.

[RFC7034] Ross, D. and T. Gondrom, "HTTP Header Field X-Frame-Options", RFC 7034, DOI 10.17487/RFC7034, October 2013, <http://www.rfc-editor.org/info/rfc7034>.

[samedomain-cookies] Goodwin, M. and J. Walker, "SameDomain Cookie Flag", 2011, <http://people.mozilla.org/~mgoodwin/SameDomain/samedomain-latest.txt>.

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

Appendix A. Acknowledgements

The same-site cookie concept documented here is indebited to Mark Goodwin's and Joe Walker's [samedomain-cookies]. Michal Zalewski, Artur Janc, Ryan Sleevi, and Adam Barth provided particularly valuable feedback on this document.

Authors' Addresses

Mike West
Google, Inc

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

Mark Goodwin
Mozilla

Email: mgoodwin@mozilla.com
URI: https://www.computerist.org/