Network Working Group M. West
Internet-Draft Google
Intended status: Standards Track March 28, 2019
Expires: September 29, 2019

HTTP State Tokens
draft-west-http-state-tokens-00

Abstract

This document describes a mechanism which allows HTTP servers to maintain stateful sessions with HTTP user agents. It aims to address some of the security and privacy considerations which have been identified in existing state management mechanisms, providing developers with a well-lit path towards our current understanding of best practice.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This document defines a state-management mechanism for HTTP that allows clients to create and persist origin-bound session identifiers that can be delivered to servers in order to enable stateful interaction. In a nutshell, each user agent will generate a single token per secure origin, and will deliver it as a "Sec-Http-State"

structured header along with requests to that origin (defined in Section 4.1 and Section 5).

Servers can configure this token's characteristics via a "Sec-Http-State-Options" response header (defined in Section 4.2 and Section 6).

That's it.

1.1. **Wait.** Don't we have cookies?

Cookies [RFC6265] are indeed a pervasive HTTP state management mechanism, and they enable practically everything interesting on the web today. That said, cookies have some issues: they're hard to use securely, they add substantial weight to users' outgoing requests, and they enable tracking users' activity across the web in potentially surprising ways.

The mechanism proposed in this document aims at a more minimal and opinionated construct which takes inspiration from some of cookies' optional characteristics. In particular:

1. The client controls the token's value, not the server.

2. The token will only be available to the network layer, not to JavaScript (including network-like JavaScript, such as Service Workers).

3. The user agent will generate only one token per origin, and will only expose the token to the origin for which it was generated.

4. Tokens will not be generated for, or delivered to, non-secure origins.

5. Tokens will be delivered only along with same-site requests by default, and can only be created from same-site contexts.

6. Each token persists for one hour after generation by default. This default expiration time can be overwritten by servers, and tokens can be reset at any time by servers, users, or user agents.

These distinctions might not be appropriate for all use cases, but seem like a reasonable set of defaults. For folks for whom these defaults aren't good enough, we'll provide developers with a few control points that can be triggered via a "Sec-HTTP-State-Options" HTTP response header, described in Section 4.2.

1.2. No. Really. We have cookies already. Why do we need this new thing?

We do have cookies. And we've defined a number of extensions to cookies to blunt some of their sharper edges: the "HttpOnly" attribute, the "Secure" attribute, "SameSite", prefixes like "__Host-" and "__Secure-", and so on. Isn't that the right way forward? Shouldn't we just push developers towards these existing flags on the existing state management primitive?

This document's underlying assumption is that it's going to be easier to teach developers about a crazy new thing that's secure by default than it would be to convince them to change their "Set-Cookie" headers to include "__Host-name=value; HttpOnly; Secure; SameSite=Lax; Path=/". A new thing resets expectations in a way that vastly exceeds the impact of explanations about the the four attributes that must be used, the one attribute that must not be used, and the weird naming convention that ought to be adopted.

Moreover, it appears that we're collectively pretty bad at helping developers understand the risks that might lead them to adopt The Good Cookie Syntax(tm) above. Adoption of these features has been quite slow. Based on data gathered from Chrome's telemetry in March, 2019, cookies are set as follows:

- ~6.8% of cookies are set with "HttpOnly".
- ~5.5% are set with "Secure".
- ~3.1% are set with "HttpOnly; Secure".
- ~0.06% are set with "SameSite=*; Secure".
- ~0.05% are set with "SameSite=*".
- ~0.03% are set with "HttpOnly; Secure; SameSite=*".
- ~0.006% are set with "SameSite=*; HttpOnly".
- ~0.005% are set with a "__Secure-" prefix.
- ~0.01% are set with a "__Host-" prefix.

In total:

- ~9.9% of cookies are marked as "HttpOnly".
- ~8.8% of cookies are marked as "Secure".

o ~0.1% of cookies are marked as "SameSite".

o ~84.2% of cookies use none of these features.

Given that "Secure" has been around since at least 1997 [RFC2109]; ~9% adoption after more than two decades is not inspiring.

1.3. Examples

User agents can deliver HTTP state tokens to a server in a "Sec-Http-State" header. For example, if a user agent has generated a token bound to "https://example.com/" whose base64 encoding is "hB2RfWaGyNk60sjHze5DzGYjSnL7tRF2HWSBx6J1o4k=" ([RFC4648], Section 4), then it would generate the following header when delivering the token along with requests to "https://example.com/":

```
              Sec-Http-State: token=*hB2RfWa...GyNko4k=*
```

The server can control certain aspects of the token's delivery by responding to requests with a "Sec-Http-State-Options" header:

```
   Sec-Http-State-Options: max-age=3600, key=*b7kuUkp...lkRioC2=*
```

2. Conventions

2.1. Conformance

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

2.2. Syntax

This document defines two Structured Headers [I-D.ietf-httpbis-header-structure]. In doing so it relies upon the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] and the OWS rule from [RFC7230].

3. Infrastructure

3.1. HTTP State Tokens

An HTTP State Token holds a session identifier which allows a user agent to maintain a stateful session with a specific origin, along with associated metadata:

- "creation" is a timestamp representing the point in time when the token was created.

- "delivery" specifies the initiating contexts from which the token can be delivered. It is an enum of either "same-origin", "same-site", or "cross-site". Unless otherwise specified, its value is "same-site".

- "key" is a server-provided key which can be used to sign requests with which the token is delivered. It is either null, or contains up to 256-bits of binary data. Unless otherwise specified, its value is null.

- "max-age" is a timestamp representing the token's lifetime in seconds. Unless otherwise specified, HTTP State Tokens have a 3600 second (1 hour) "max-age".

- "value" is the token's value (surprising, right?). It contains up to 256-bits of binary data.

An HTTP State Token is said to be "expired" if its "creation" timestamp plus "max-age" seconds is in the past.

3.2. Requests and Responses

This document relies upon the definitions of "request" and "response" found in [Fetch].

A request's delivery scope can be obtained as follows:

1. Let "request-origin" be the request's "origin", and "target-origin" be the request's "URL"'s "origin".

2. If the request was generated by the user agent as a response to direct user interaction with the user agent (e.g. the user typed an address into the agent's address bar, clicked a bookmark, or etc.), return "same-origin".

3. If "request-origin" is same-origin with "target-origin", return "same-origin".

4. If "request-origin"'s registrable domain is the same as "target-origin"'s registrable domain, return "same-site".

5. Return "cross-site".

3.3. Token Storage

User agents MUST keep a list of all the unexpired HTTP State Tokens which have been created. For the purposes of this document, we'll assume that user agents keep this list in the form of a map whose keys are origins, and whose values are HTTP State Tokens.

This map exposes three functions:

- o An HTTP State Token can be stored for a given origin. If the origin already exists in the map, the entry's value will be overwritten with the new HTTP State Token.

- o An origin's HTTP State Token can be retrieved. If the origin does not exist in the map, "null" will be returned instead.

- o An origin (along with its HTTP State Token) can be deleted from the map.

The map is initially empty.

3.3.1. Generate an HTTP State Token for an origin

The user agent can generate a new HTTP State Token for an origin using an algorithm equivalent to the following:

1. Delete "origin" from the user agent's token store.

2. Let "token" be a newly created HTTP State Token with its properties set as follows:

   * "creation": The current time.

   * "delivery": "same-site"

   * "key": null

   * "max-age": 3600

   * "value": 256 cryptographically random bits.

3. Store "token" in the user agent's token store for "origin".

4. If the user agent has defined a "NotifyHostHTTPStateReset()" algorithm, call it with "origin".

5. Return "token".

Note: Step 4 recognizes that user agents may wish to notify an origin's developers that HTTP state has been reset in order to enable cleanup of state stored client-side. HTML might, for instance, wish to post a message to a specially-named "BroadcastChannel" to enable this kind of work. This could take something like the following form:

```
let resetChannel = new BroadcastChannel('http-state-reset'));
resetChannel.onmessage = e => { /* Do exciting cleanup here. */ };
```

4. Syntax

4.1. The 'Sec-Http-State' HTTP Header Field

The "Sec-Http-State" HTTP header field allows user agents to deliver HTTP state tokens to servers as part of an HTTP request.

"Sec-Http-State" is a Structured Header [I-D.ietf-httpbis-header-structure]. Its value MUST be a dictionary ([I-D.ietf-httpbis-header-structure], Section 3.1). Its ABNF is:

```
Sec-Http-State = sh-dictionary
```

The dictionary MUST contain:

- Exactly one member whose key is "token", and whose value is binary content ([I-D.ietf-httpbis-header-structure], Section 3.9) that encodes the HTTP state token's value for the origin to which the header is delivered.

  If the "token" member contains more than 256 bits of binary content, the member MUST be ignored.

The dictionary MAY contain:

- Exactly one member whose key is "sig", and whose value is binary content ([I-D.ietf-httpbis-header-structure], Section 3.9) that encodes a signature over the token and the request which contains it, using a key previously delivered by the server. This mechanism is described in Section 5.2.

  If the "sig" member contains more than 256 bits of binary content, the member MUST be ignored.

The "Sec-Http-State" header is parsed per the algorithm in Section 4.2 of [I-D.ietf-httpbis-header-structure]. Servers MUST ignore the header if parsing fails, or if the parsed header does not contain a member whose key is "token".

User agents will attach a "Sec-Http-State" header to outgoing requests according to the processing rules described in Section 5.

4.2. The 'Sec-Http-State-Options' HTTP Header Field

The "Sec-Http-State-Options" HTTP header field allows servers to deliver configuration information to user agents as part of an HTTP response.

"Sec-Http-State-Options" is a Structured Header [I-D.ietf-httpbis-header-structure]. Its value MUST be a dictionary ([I-D.ietf-httpbis-header-structure], Section 3.1). Its ABNF is:

```
               Sec-Http-State-Options = sh-dictionary
```

The "Sec-Http-State-Options" header is parsed per the algorithm in Section 4.2 of [I-D.ietf-httpbis-header-structure]. User agents MUST ignore the header if parsing fails.

The dictionary MAY contain:

- Exactly one member whose key is "key", and whose value is binary content ([I-D.ietf-httpbis-header-structure], Section 3.10) that encodes an key which can be used to generate a signature over outgoing requests.

- Exactly one member whose key is "delivery", and whose value is one of the following tokens ([I-D.ietf-httpbis-header-structure], Section 3.9): "same-origin", "same-site", or "cross-site".

  If the "delivery" member contains an unknown identifier, the member MUST be ignored.

- Exactly one member whose key is "max-age", and whose value is an integer ([I-D.ietf-httpbis-header-structure], Section 3.6) representing the server's desired lifetime for its HTTP State Token.

  If the "max-age" member contains anything other than a positive integer, the member MUST be ignored.

User agents will process the "Sec-Http-State-Options" header on incoming responses according to the processing rules described in Section 6.

4.2.1. Examples

4.2.1.1. Cross-Site Delivery

Some servers will require access to their tokens from cross-site contexts (perhaps to support authenticated activity or single-sign on, etc). These servers can request a "cross-site" delivery option by delivering the following header:

```
          Sec-Http-State-Options: delivery=cross-site, ...
```

4.2.1.2. Token Lifetime

Other servers might want their sessions to persist for more than an hour. These servers can request a more reasonable token lifetime lifetime by by delivering the following header:

```
            Sec-Http-State-Options: max-age=2592000, ...
```

Servers may also wish to explicitly trigger the token's expiration (upon signout, for instance). Setting a "max-age" of "0" does the trick:

```
               Sec-Http-State-Options: max-age=0, ...
```

4.2.1.3. Token Provenance

For some servers, the client-generated token will be enough to maintain state. They can treat it as an opaque session identifier, and bind the user's state to it server-side. Other servers will require additional assurance that they can trust the token's provenance. To that end, servers can generate a unique key, associate it with the session identifier on the server, and deliver it to the client via an HTTP response header:

```
     Sec-Http-State-Options: key=*ZH0GxtBMWA...nJudhZ8dtz*, ...
```

Clients will store that key, and use it to generate a signature over some set of data that mitigates the risk of token capture:

```
            Sec-HTTP-State:
                token=*J6BRKa...MonM*,
                sig=*(HMAC-SHA265(key, token+metadata))*
```

Note: This part in particular is not fully baked, and we need to do some more work to flesh out the threat model (see also Token Binding). Look at it as an area to explore, not a solidly thought-out solution.

5. Delivering HTTP State Tokens

User agents deliver HTTP state tokens to servers by appending a "Sec-Http-State" header field to outgoing requests.

This specification provides algorithms which are called at the appropriate points in [Fetch] in order to attach "Sec-Http-State" headers to outgoing requests, and to ensure that "Sec-Http-State-Options" headers are correctly processed.

5.1. Attach HTTP State Tokens to a request

The user agent can attach HTTP State Tokens to a given request using an algorithm equivalent to the following. This algorithm is intended to execute as the request is being sent out over the network (after Service Worker processing), perhaps after the "Cookie" header is handled in step 5.17.1 of Section 4.5 of [Fetch], describing the "HTTP-network-or-cache fetch" algorithm:

1. If the user agent is configured to suppress explicit identifiers for the request, or if the request's URL is not _a priori_ authenticated [Mixed-Content], then skip the remaining steps in this algorithm, and return without modifying the request.

2. Let "target-origin" be the origin of "request"'s current URL.

3. Let "request-token" be the result of retrieving origin's token from the user agent's token store, or "null" if no such token exists.

4. If "request-token" is expired, clear the user agent's token store for "target-origin", and set "request-token" to "null".

5. If "request-token" is "null", then:

    1. If "request"'s delivery scope is "cross-site", return without modifying the request.

        Note: As the default "delivery" for HTTP State Tokens is "same-site", we return early rather than generating a token for a cross-site request.

    2. Set "request-token" to the result of generating an HTTP State Token for "target-origin", as defined in Section 3.3.1.

6. Return without modifying the request if either of the following statements are true:

  * "request-token"'s "delivery" is "same-origin", and "request"'s delivery scope is not "same-origin".

  * "request-token"'s "delivery" is "same-site", and "request"'s delivery scope is neither "same-origin" nor "same-site".

7. Let "serialized-value" be the base64 encoding ([RFC4648], Section 4) of "request-token"'s value.

8. Insert a member into "header-value" whose key is "token" and whose value is "serialized-value".

9. If "request-token"'s "key" is not null, then insert a member into "header-value" whose key is "sig", and whose value is the result of executing Section 5.2 on request, "serialized-value", and "request-token"'s "key".

10. Append a header to "request"'s header list whose name is "Sec-Http-State", and whose value is the result of serializing "header-value" ([I-D.ietf-httpbis-header-structure], Section 4.1).

5.2. Generate a request's signature

If the origin server provides a "key", the user agent will use it to sign any outgoing requests which target that origin and include an HTTP State Token. Note that the signature is produced before adding the "Sec-Http-State" header to the request.

Given a request, a base64-encoded token value, and a key:

1. Let "cbor-request" be the result of building a CBOR representation [RFC7409] of the given request, as specified in the first element of the array described in Section 3.2 of [I-D.yasskin-http-origin-signed-responses].

2. Add an item to "cbor-request" which maps the byte string ':token' to the byte string containing the given base64-encoded token value.

3. Return the result of computing HMAC-SHA256 [RFC2104] over the canonical CBOR serialization of "cbor-request" (Section 3.4 of [I-D.yasskin-http-origin-signed-responses]), using the given "key".

5.2.1. Example

The following request:

```
GET / HTTP/1.1
Host: example.com
Accept: */*
```

results in the following CBOR representation (represented using the extended diagnostic notation from Appendix G of [I-D.ietf-cbor-cddl]):

```
{
  ':method': 'GET',
  ':token': 'hB2RfWaGyNk60sjHze5DzGYjSnL7tRF2HWSBx6J1o4k='
  ':url': 'https://example.com/',
  'accept': '*/*',
}
```

6. Configuring HTTP State Tokens

Servers configure the HTTP State Token representing a given users' state by appending a "Sec-Http-State-Options" header field to outgoing responses.

User agents MUST process this header on a given response as per the following algorithm, which is intended to be called after the "Set-Cookie" header is handled in step 11.4 of Section 4.6 of [Fetch], which defines the "HTTP-network fetch" algorithm.

1. Let "response-origin" be the origin of response's URL.

2. If the response's URL is not _a priori_ authenticated [Mixed-Content], return without altering "response-origin"'s HTTP State Token.

3. Let "token" be the result of retrieving "response-origin"'s token from the user agent's token store, or "null" if no such token exists.

4. If "token" is expired, clear the user agent's token store for "response-origin", and set "token" to "null".

5. If "token" is "null", then:

    1. If "request"'s delivery scope is "cross-site", return without modifying the request.

Note: As the default "delivery" for HTTP State Tokens is "same-site", we return early rather than generating a token for a cross-site request.

2. Set "token" to the result of generating an HTTP State Token for "target-origin", as defined in Section 3.3.1.

6. If the response's header list contains "Sec-Http-State-Options", then:

   1. Let "header" be the result of getting response's "Sec-Http-State-Options" header, and parsing parsing it per the algorithm in Section 4.2 of [I-D.ietf-httpbis-header-structure].

   2. Return without altering "response-origin"'s HTTP State Token if any of the following conditions hold:

      + Parsing the header results in failure.

      + "header" has a member named "key" whose value is not a byte sequence (Section 3.10 of [I-D.ietf-httpbis-header-structure])

      + "header" has a member named "delivery" whose value is not one of the following tokens (Section 3.9 of [I-D.ietf-httpbis-header-structure]): "same-origin", "same-site", and "cross-site".

      + "header" has a member named "max-age" whose value is not a positive integer (Section 3.6 of [I-D.ietf-httpbis-header-structure]).

   3. If "header" has a member named "key", set "token"'s "key" to the member's value.

   4. If "header" has a member named "delivery", set "token"'s "delivery" to the member's value.

   5. If "header" has a member named "max-age":

      1. If the member's value is "0", generate a new HTTP State Token for "response-origin" as defined in Section 3.3.1.

         Otherwise, set "token"'s "max-age" to the member's value.

Note that "max-age" is processed last, meaning that any other options specified alongside "max-age=0" will be de facto ignored as a new token is generated, replacing the old.

7. Security and Privacy Considerations

HTTP State Tokens aim to mitigate some of the security and privacy drawbacks that decades of implementation experience with cookies have laid bare. It would be worthwhile to skim through the privacy considerations (Section 7 of [RFC6265]) and security considerations (Section 8 of [RFC6265]) of that existing state management mechanism, as it forms a foundation upon which this document builds.

7.1. Confidentiality and Integrity

HTTP State Tokens improve upon cookies' weak confidentiality/ integrity guarantees (see Sections 8.3, 8.5, 8.6, and 8.7 of [RFC6265]) in several ways:

1. User agents MUST require secure channels (such as TLS) for delivery and configuration of HTTP State Tokens. User agents cannot be induced to deliver an origin's tokens across channels visible to (and modifiable by) network attackers, nor can an attack on DNS cause tokens to be revealed (as any server to which the user could be directed will also need to authenticate itself, which is presumably difficult).

2. HTTP State Tokens are mapped to origins, matching developers expectations for client-side data generally. This ensures that tokens are isolated by host and port: code running on "https://bar.example.com/" cannot alter state on "https://foo.example.com/" without the latter's cooperation, and that the same applies to "https://example.com:8000/" and "https://example.com:80/".

   Note that this origin binding means that there are no path restrictions for tokens. Servers relying upon these tokens for state management SHOULD NOT run mutually distrusting services on different paths of the same origin.

3. User agents MUST NOT expose HTTP State Tokens to non-HTTP APIs which are web-accessible, thereby reducing the risk of accidental exposure via cross-site scripting attack.

   Further, the "Sec-" prefix on both "Sec-HTTP-State" and "Sec-HTTP-State-Options" ensures that both are considered "forbidden header names" by [Fetch]. The latter should also be treated as a "forbidden response header".

7.2. Signed Sessions

HTTP State Tokens embrace the session identifier pattern discussed in Section 8.4 of [RFC6265] by requiring that the client control the token's value, setting it to a fixed-length, random byte sequence. The client's control mitigates the risk of sensitive information being stored in the token directly, and the token's length makes it unlikely to be easily guessed.

Some servers will be interested in proving the token's provenance over time, which they do today by storing cookies with signed values. Since storing a signed value directly is impossible in a client-controlled world, servers can instead store a "key", which is used to sign outgoing requests. Since this key is never exposed directly to the web, it provides a reasonable guarantee of client stability over time which a server can rely upon when making risk judgements.

7.3. User Control

User agents MUST provide users with the ability to control the creation and distribution of HTTP State Tokens, just as they do for cookies today. This certainly means providing controls over first- vs third-party distribution, control over the origins which can store state, control over the state presented to origins, visibility into the state of the user agent's token store, and etc.

Further, this document grants user agents wide latitude to experiment with various distribution policies and limitations. The capabilities offered by "delivery" and "max-age" should be considered upper bounds on distribution, within which user agents are free to roam.

7.4. Lifetime

By default, HTTP State Tokens live for an hour, which is a compromise between the reasonable desire of servers to maintain state across a given user's session, and the privacy risks associated with long-lived tokens stored on a user's disk.

Servers that desire a longer session lifetime can explicitly request an extension, which the browser can choose to act on.

7.5. Ambient Authority and Cross-Site Delivery

HTTP State Tokens, like cookies, provide a form of ambient authority (see Section 8.2 of [RFC6265]). By default, this authority is limited to requests initiated by same-site actors, which serves as a reasonable mitigation against some classes of attack (e.g.

"https://evil.com/" making authenticated requests to "https://example.com/").

Servers that desire to interact in an authenticated manner in cross-site contexts are required to opt-into doing so by delivering an appropriate "delivery" value in a "Sec-HTTP-State-Options" response header. Servers which choose to do so SHOULD take reasonable precautions, implementing CSRF tokens for sensitive actions, and taking stock of the context from which a given request is initiated (by examining incoming "Referrer", "Origin", and "Sec-Fetch-Site" headers).

Further, tokens can only be created in same-origin or same-site contexts, which means that cross-site identifier would only be available after the relevant origin was visited in a same-site context, and explicitly declared its tokens as being deliverable cross-site (at which point the user agent is empowered to make some decisions about how to handle that declaration).

8. IANA Considerations

8.1. Header Field Registry

This document registers the "Sec-Http-State" and "Sec-Http-State-Options" header fields in the "Permanent Message Header Field Names" registry located at https://www.iana.org/assignments/message-headers [1].

8.1.1. Sec-Http-State Header Field

Header field name: Sec-Http-State

Applicable protocol: http

Status: experimental

Author/Change controller: IETF

Specification document(s): This document (see Section 4.1)

Related information: (empty)

8.1.2. Sec-Http-State-Options Header Field

Header field name: Sec-Http-State-Options

Applicable protocol: http

Status: experimental

Author/Change controller: IETF

Specification document(s): This document (see Section 4.2)

Related information: (empty)

9. References

9.1. Normative References

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

[I-D.ietf-httpbis-header-structure] Nottingham, M. and P. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-09 (work in progress), December 2018.

[I-D.yasskin-http-origin-signed-responses] Yasskin, J., "Signed HTTP Exchanges", draft-yasskin-http-origin-signed-responses-05 (work in progress), January 2019.

[Mixed-Content] West, M., "Mixed Content", n.d., <https://w3c.github.io/webappsec-mixed-content/>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

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

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

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

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

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

[RFC7409] Haleplidis, E. and J. Halpern, "Forwarding and Control Element Separation (ForCES) Packet Parallelization", RFC 7409, DOI 10.17487/RFC7409, November 2014, <https://www.rfc-editor.org/info/rfc7409>.

[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>.

9.2. Informative References

[I-D.abarth-cake]
Barth, A., "Origin Cookies", draft-abarth-cake-01 (work in progress), March 2011.

[I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR and JSON data structures", draft-ietf-cbor-cddl-08 (work in progress), March 2019.

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

9.3. URIs

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

Appendix A. Acknowledgements

This document owes much to Adam Barth's [I-D.abarth-cake] and [RFC6265].

Appendix B. Changes

RFC Editor: Please remove this section before publication.

B.1. Since the beginning of time

o This document was created.

Author's Address

Mike West
Google

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