Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 29, 2020

J. Yasskin
Google

September 26, 2019

Bundled HTTP Exchanges
draft-yasskin-wpack-bundled-exchanges-02

Abstract

Bundled exchanges provide a way to bundle up groups of HTTP request+response pairs to transmit or store them together. They can include multiple top-level resources with one identified as the default by a manifest, provide random access to their component exchanges, and efficiently store 8-bit resources.

Note to Readers

Discussion of this draft takes place on the wpack mailing list (wpack@ietf.org), which is archived at https://www.ietf.org/mailman/listinfo/wpack [1].

The source code and issues list for this draft can be found in https://github.com/WICG/webpackage [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 March 29, 2020.

Copyright Notice

Table of Contents

1. Introduction

To satisfy the use cases in [I-D.yasskin-webpackage-use-cases], this document proposes a new bundling format to group HTTP resources. Several of the use cases require the resources to be signed: that's provided by bundling signed exchanges ([I-D.yasskin-http-origin-signed-responses]) rather than natively in this format.

1.1. Terminology

Exchange (noun) An HTTP request+response pair. This can either be a request from a client and the matching response from a server or the request in a PUSH_PROMISE and its matching response stream. Defined by Section 8 of [RFC7540].

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

1.2. Mode of specification

This specification defines how conformant bundle parsers work. It does not constrain how encoders produce a bundle: although there are some guidelines in Section 4, encoders MAY produce any sequence of bytes that a conformant parser would parse into the intended semantics.

This specification uses the conventions and terminology defined in the Infra Standard ([INFRA]).

2. Semantics

A bundle is logically a set of HTTP exchanges, with a URL identifying the manifest(s) of the bundle itself.

While the order of the exchanges is not semantically meaningful, it can significantly affect performance when the bundle is loaded from a network stream.

A bundle is parsed from a stream of bytes, which is assumed to have the attributes and operations described in Section 2.1.

Bundle parsers support two operations, Load a bundle's metadata (Section 2.2) and Load a response from a bundle (Section 2.3) each of which can return an error instead of their normal result.

A client is expected to load the metadata for a bundle as soon as it starts downloading it or otherwise discovers it. Then, when fetching ([FETCH]) a request, the client is expected to match it against the requests in the metadata, and if one matches, load that request's response.

2.1. Stream attributes and operations

- A sequence of *available bytes*. As the stream delivers bytes, these are appended to the available bytes.
- An *EOF* flag that's true if the available bytes include the entire stream.
- A *current offset* within the available bytes.
- A *seek to offset N* operation to set the current offset to N bytes past the beginning of the available bytes. A seek past the end of the available bytes blocks until N bytes are available. If the stream ends before enough bytes are received, either due to a network error or because the stream has a finite length, the seek fails.
- A *read N bytes* operation, which blocks until N bytes are available past the current offset, and then returns them and seeks forward by N bytes. If the stream ends before enough bytes are received, either due to a network error or because the stream has a finite length, the read operation returns an error instead.

2.2. Load a bundle's metadata

This takes the bundle's stream and returns either an error (where an error is a "format error" or a "version error"), an error with a fallback URL (which is also the primaryUrl when the bundle parses successfully), or a map ([INFRA]) of metadata containing at least keys named:

primaryUrl The URL of the main resource in the bundle. If the client can't process the bundle for any reason, this is also the fallback URL, a reasonable URL to try to load instead.

requests A map ([INFRA]) whose keys are URLs and whose values consist of either:

* A single "ResponseMetadata" value for a non-content-negotiated resource or

* A set of content-negotiated resources represented by

  + A "Variants" header field value ([I-D.ietf-httpbis-variants]) and

  + A map ([INFRA]) from each of the possible combinations of one available-value for each variant-axis to a "ResponseMetadata" structure. Load a response from a bundle can use the "ResponseMetadata" structures to find the matching response.

manifest The URL of the bundle's manifest(s). This is a URL to support bundles with multiple different manifests, where the client uses content negotiation to select the most appropriate one.

The map may include other items added by sections defined in the Web Bundle Section Name Registry.

This operation only waits for a prefix of the stream that, if the bundle is encoded with the "responses" section last, ends before the first response.

This operation's implementation is in Section 3.3.

2.2.1. Load a bundle's metadata from the end

If a bundle's bytes are embedded in a longer sequence rather than being streamed, a parser can also load them starting from a pointer to the last byte of the bundle. This returns the same data as Section 2.2.

This operation's implementation is in Section 3.3.6.

2.3. Load a response from a bundle

This takes the stream of bytes representing the bundle, a request ([FETCH]), and the "ResponseMetadata" returned from Section 2.2 for the appropriate content-negotiated resource within the request's URL, and returns the response ([FETCH]) matching that request.

This operation can be completed without inspecting bytes other than those that make up the loaded response, although higher-level operations like proving that an exchange is correctly signed

([I-D.yasskin-http-origin-signed-responses]) may need to load other responses.

A client will generally want to load the response for a request that the client generated. For a URL with multiple variants, the client SHOULD use the algorithm in Section 4 of [I-D.ietf-httpbis-variants] to select the best variant.

This operation's implementation is in Section 3.4.

3. Format

3.1. Top-level structure

This section is non-normative.

A bundle holds a series of named sections. The beginning of the bundle maps section names to the range of bytes holding that section. The most important section is the "index" (Section 3.3.1), which similarly maps serialized HTTP requests to the range of bytes holding that request's serialized response. Byte ranges are represented using an offset from some point in the bundle _after_ the encoding of the range itself, to reduce the amount of work needed to use the shortest possible encoding of the range.

Future specifications can define new sections with extra data, and if necessary, these sections can be marked "critical" (Section 3.3.4) to prevent older parsers from using the rest of the bundle incorrectly.

The bundle is a CBOR item ([CBORbis]) with the following CDDL ([CDDL]) schema:

```
webbundle = [
  ; &#127760;&#128230; in UTF-8.
  magic: h'F0 9F 8C 90 F0 9F 93 A6',
  version: bytes .size 4,
  primary-url: whatwg-url,
  section-lengths: bytes .cbor [* (section-name: tstr, length: uint) ],
  sections: [* any ],
  length: bytes .size 8,  ; Big-endian number of bytes in the bundle.
]

$section-name /= "index" / "manifest" / "signatures" / "critical" / "responses"

$section /= index / manifest / signatures / critical / responses

responses = [*response]

whatwg-url = tstr
```

3.2. Serving constraints

When served over HTTP, a response containing an "application/webbundle" payload MUST include at least the following response header fields, to reduce content sniffing vulnerabilities (Section 5.2):

- Content-Type: application/webbundle
- X-Content-Type-Options: nosniff

3.3. Load a bundle's metadata

A bundle holds a series of sections, which can be accessed randomly using the information in the "section-lengths" CBOR item, which holds a list of alternating section names and section lengths:

```
section-lengths = [* (section-name: tstr, length: uint) ],
```

To implement Section 2.2, the parser MUST run the following steps, taking the "stream" as input.

1. Seek to offset 0 in "stream". Assert: this operation doesn't fail.
2. If reading 10 bytes from "stream" returns an error or doesn't return the bytes with hex encoding "86 48 F0 9F 8C 90 F0 9F 93 A6" (the CBOR encoding of the 6-item array initial byte and

8-byte bytestring initial byte, followed by 🌐📦 in UTF-8), return a "format error".

3. Let "version" be the result of reading 5 bytes from "stream". If this is an error, return a "format error".

4. Let "urlType" and "urlLength" be the result of reading the type and argument of a CBOR item from "stream" (Section 3.5.3). If this is an error or "urlType" is not 3 (a CBOR text string), return a "format error".

5. Let "fallbackUrlBytes" be the result of reading "urlLength" bytes from "stream". If this is an error, return a "format error".

6. Let "fallbackUrl" be the result of parsing ([URL]) the UTF-8 decoding of "fallbackUrlBytes" with no base URL. If either the UTF-8 decoding or parsing fails, return a "format error".

   Note: From this point forward, errors also include the fallback URL to help clients recover.

7. If "version" does not have the hex encoding "44 31 00 00 00" (the CBOR encoding of a 4-byte byte string holding an ASCII "1" followed by three 0 bytes), return a "version error" with "fallbackUrl".

   Note: RFC EDITOR PLEASE DELETE THIS NOTE; Implementations of drafts of this specification MUST NOT use the version "1" in this byte string, and MUST instead define an implementation-specific string to identify which draft is implemented. This string SHOULD match the version used in the draft's MIME type (Section 6.1).

8. Let "sectionLengthsLength" be the result of getting the length of the CBOR bytestring header from "stream" (Section 3.5.2). If this is an error, return a "format error" with "fallbackUrl".

9. If "sectionLengthsLength" is 8192 (8*1024) or greater, return a "format error" with "fallbackUrl".

10. Let "sectionLengthsBytes" be the result of reading "sectionLengthsLength" bytes from "stream". If "sectionLengthsBytes" is an error, return a "format error" with "fallbackUrl".

11. Let "sectionLengths" be the result of parsing one CBOR item (Section 3.5) from "sectionLengthsBytes", matching the section-

lengths rule in the CDDL ([CDDL]) above. If "sectionLengths" is an error, return a "format error" with "fallbackUrl".

12. Let ("sectionsType", "numSections") be the result of parsing the type and argument of a CBOR item from "stream" (Section 3.5.3).

13. If "sectionsType" is not "4" (a CBOR array) or "numSections" is not half of the length of "sectionLengths", return a "format error" with "fallbackUrl".

14. Let "sectionsStart" be the current offset within "stream".

    For example, if "sectionLengthsLength" were 52 and "sectionLengths" contained 4 items (2 sections), "sectionsStart" would be 65 (10 initial bytes + a 2-byte bytestring header to describe a 52-byte bytestring + 52 bytes of section lengths + a 1-byte array header for the 2 sections).

15. Let "knownSections" be the subset of the Section 6.2 that this client has implemented.

16. Let "ignoredSections" be an empty set.

17. Let "sectionOffsets" be an empty map ([INFRA]) from section names to (offset, length) pairs. These offsets are relative to the start of "stream".

18. Let "currentOffset" be "sectionsStart".

19. For each (""name"", "length") pair of adjacent elements in "sectionLengths":

    1. If ""name""'s specification in "knownSections" says not to process other sections, add those sections' names to "ignoredSections".

       Note: The "ignoredSections" enables sections that supercede other sections to be introduced in the future. Implementations that don't implement any such sections are free to omit the relevant steps.

    2. If "sectionOffsets["name"]" exists, return a "format error" with "fallbackUrl". That is, duplicate sections are forbidden.

    3. Set "sectionOffsets["name"]" to ("currentOffset", "length").

    4. Set "currentOffset" to "currentOffset + length".

20. If the "responses" section is not last in "sectionLengths", return a "format error" with "fallbackUrl". This allows a streaming parser to assume that it'll know the requests by the time their responses arrive.

21. Let "metadata" be a map ([INFRA]) initially containing the single key/value pair ""primaryUrl""/"fallbackUrl".

22. For each ""name"" --> ("offset", "length") triple in "sectionOffsets":

    1. If ""name"" isn't in "knownSections", continue to the next triple.

    2. If ""name""'s Metadata field (Section 6.2) is "No", continue to the next triple.

    3. If ""name"" is in "ignoredSections", continue to the next triple.

    4. Seek to offset "offset" in "stream". If this fails, return a "format error" with "fallbackUrl".

    5. Let "sectionContents" be the result of reading "length" bytes from "stream". If "sectionContents" is an error, return a "format error" with "fallbackUrl".

    6. Follow ""name""'s specification from "knownSections" to process the section, passing "sectionContents", "stream", "sectionOffsets", and "metadata". If this returns an error, return a "format error" with "fallbackUrl".

23. Assert: "metadata" has an entry with the key "primaryUrl".

24. If "metadata" doesn't have entries with keys "requests" and "manifest", return a "format error" with "fallbackUrl".

25. Return "metadata".

3.3.1. Parsing the index section

The "index" section defines the set of HTTP requests in the bundle and identifies their locations in the "responses" section. It consists of a map from URL strings to arrays consisting of a "Variants" header field value ([I-D.ietf-httpbis-variants]) followed by one "location-in-responses" pair for each of the possible combinations of available-values within the "Variants" value in lexicographic (row-major) order.

For example, given a "variants-value" of "Accept-Encoding;gzip;br, Accept-Language;en;fr;ja", the list of "location-in-responses" pairs will correspond to the "VariantKey"s:

- gzip;en
- gzip;fr
- gzip;ja
- br;en
- br;fr
- br;ja

The order of variant-axes is important. If the "variants-value" were "Accept-Language;en;fr;ja, Accept-Encoding;gzip;br" instead, the "location-in-responses" pairs would instead correspond to:

- en;gzip
- en;br
- fr;gzip
- fr;br
- ja;gzip
- ja;br

As a special case, an empty "variants-value" indicates that there is only one resource at the specified URL and that no content negotiation is performed.

```
index = {* whatwg-url => [ variants-value, +location-in-responses ] }
variants-value = bstr
location-in-responses = (offset: uint, length: uint)
```

A "ResponseMetadata" struct identifies a byte range within the bundle stream, defined by an integer offset from the start of the stream and the integer number of bytes in the range.

To parse the index section, given its "sectionContents", the "sectionOffsets" map, and the "metadata" map to fill in, the parser MUST do the following:

1. Let "index" be the result of parsing "sectionContents" as a CBOR item matching the "index" rule in the above CDDL (Section 3.5). If "index" is an error, return an error.

2. Let "requests" be an initially-empty map ([INFRA]) from URLs to response descriptions, each of which is either a single "location-in-stream" value or a pair of a "Variants" header field value ([I-D.ietf-httpbis-variants]) and a map from that value's possible "Variant-Key"s to "location-in-stream" values, as described in Section 2.2.

3. Let "MakeRelativeToStream" be a function that takes a "location-in-responses" value ("offset", "length") and returns a "ResponseMetadata" struct or error by running the following sub-steps:

    1. If "offset" + "length" is larger than "sectionOffsets["responses"].length", return an error.

    2. Otherwise, return a "ResponseMetadata" struct whose offset is "sectionOffsets["responses"].offset" + "offset" and whose length is "length".

4. For each ("url", "responses") entry in the "index" map:

    1. Let "parsedUrl" be the result of parsing ([URL]) "url" with no base URL.

    2. If "parsedUrl" is a failure, its fragment is not null, or it includes credentials, return an error.

    3. If the first element of "responses" is the empty string:

        1. If the length of "responses" is not 3 (i.e. there is more than one "location-in-responses" in responses), return an error.

        2. Otherwise, assert that "requests"["parsedUrl"] does not exist, and set "requests"["parsedUrl"] to "MakeRelativeToStream(location-in-responses)", where "location-in-responses" is the second and third elements of "responses". If that returns an error, return an error.

    4. Otherwise:

        1. Let "variants" be the result of parsing the first element of "responses" as the value of the "Variants" HTTP header

field (Section 2 of [I-D.ietf-httpbis-variants]). If this fails, return an error.

2. Let "variantKeys" be the Cartesian product of the lists of available-values for each variant-axis in lexicographic (row-major) order. See the examples above.

3. If the length of "responses" is not "2 * len(variantKeys) + 1", return an error.

4. Set "requests"["parsedUrl"] to a map from "variantKeys"["i"] to the result of calling "MakeRelativeToStream" on the "location-in-responses" at "responses"["2*i+1"] and "responses"["2*i+2"], for "i" in ["0", "len(variantKeys)"). If any "MakeRelativeToStream" call returns an error, return an error.

5. Set "metadata["requests"]" to "requests".

3.3.2. Parsing the manifest section

The "manifest" section records a single URL identifying the manifest of the bundle. The URL MUST refer to the one or more response(s) contained in the bundle itself.

The bundle can contain multiple resources at this URL, and the client is expected to content-negotiate for the best one. For example, a client might select the one with an "accept" header of "application/manifest+json" ([appmanifest]) and an "accept-language" header of "es-419".

manifest = whatwg-url

To parse the manifest section, given its "sectionContents" and the "metadata" map to fill in, the parser MUST do the following:

1. Let "urlString" be the result of parsing "sectionContents" as a CBOR item matching the above "manifest" rule (Section 3.5. If "urlString" is an error, return that error.

2. Let "url" be the result of parsing ([URL]) "urlString" with no base URL.

3. If "url" is a failure, its fragment is not null, or it includes credentials, return an error.

4. Set "metadata["manifest"]" to "url".

3.3.3. Parsing the signatures section

The "signatures" section vouches for the resources in the bundle.

The section can contain as many signatures as needed, each by some authority, and each covering an arbitrary subset of the resources in the bundle. Intermediates, including attackers, can remove signatures from the bundle without breaking the other signatures.

The bundle parser's client is responsible to determine the validity and meaning of each authority's signatures. In particular, the algorithm below does not check that signatures are valid. For example, a client might:

- Use the ecdsa_secp256r1_sha256 algorithm defined in Section 4.2.3 of [TLS1.3] to check the validity of any signature with an EC public key on the secp256r1 curve.
- Reject all signatures by an RSA public key.
- Treat an X.509 certificate with the CanSignHttpExchanges extension (Section 4.2 of [I-D.yasskin-http-origin-signed-responses]) and a valid chain to a trusted root as an authority that vouches for the authenticity of resources claimed to come from that certificate's domains.
- Treat an X.509 certificate with another extension or EKU as vouching that a particular analysis has run over the signed resources without finding malicious behavior.

A client might also choose different behavior for those kinds of authorities and keys.

```
signatures = [
  authorities: [*authority],
  vouched-subsets: [*{
    authority: index-in-authorities,
    sig: bstr,
    signed: bstr  ; Expected to hold a signed-subset item.
  }],
]
authority = augmented-certificate
index-in-authorities = uint

signed-subset = {
  validity-url: whatwg-url,
  auth-sha256: bstr,
  date: uint,
  expires: uint,
  subset-hashes: {+
    whatwg-url => [variants-value, +resource-integrity]
  },
  * tstr => any,
}
resource-integrity = (header-sha256: bstr, payload-integrity-header: tstr)
```

The "augmented-certificate" CDDL rule comes from Section 3.3 of [I-D.yasskin-http-origin-signed-responses].

To parse the signatures section, given its "sectionContents", the "sectionOffsets" map, and the "metadata" map to fill in, the parser MUST do the following:

1. Let "signatures" be the result of parsing "sectionContents" as a CBOR item matching the "signatures" rule in the above CDDL (Section 3.5).

2. Set "metadata["authorities"]" to the list of authorities in the first element of the "signatures" array.

3. Set "metadata["vouched-subsets"]" to the second element of the "signatures" array.

3.3.4. Parsing the critical section

The "critical" section lists sections of the bundle that the client needs to understand in order to load the bundle correctly. Other sections are assumed to be optional.

```
critical = [*tstr]
```

To parse the critical section, given its "sectionContents" and the "metadata" map to fill in, the parser MUST do the following:

1. Let "critical" be the result of parsing "sectionContents" as a CBOR item matching the above "critical" rule (Section 3.5). If "critical" is an error, return that error.

2. For each value "sectionName" in the "critical" list, if the client has not implemented sections named "sectionName", return an error.

This section does not modify the returned metadata.

3.3.5. The responses section

The responses section does not add any items to the bundle metadata map. Instead, its offset and length are used in processing the index section (Section 3.3.1).

3.3.6. Starting from the end

The length of a bundle is encoded as a big-endian integer inside a CBOR byte string at the end of the bundle.

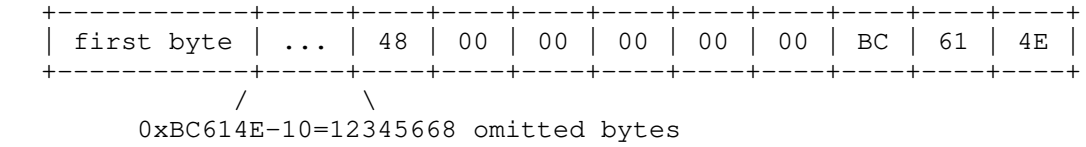

Figure 1: Example trailing bytes

Parsing from the end allows the bundle to be appended to another format such as a self-extracting executable.

To implement Section 2.2.1, taking a sequence of bytes "bytes", the client MUST:

1. Let "byteStringHeader" be "bytes[bytes.length - 9]". If "byteStringHeader is not "0x48‘ (the CBOR ([CBORbis]) initial byte for an 8-byte byte string), return an error.

2. Let "bundleLength" be "[bytes[bytes.length - 8], bytes[bytes.length])" (the last 8 bytes) interpreted as a big-endian integer.

3. If "bundleLength > bytes.length", return an error.

4. Let "stream" be a new stream whose:

   * Available bytes are "[bytes[bytes.length - bundleLength], bytes[bytes.length])".

   * EOF flag is set.

   * Current offset is initially 0.

   * The seek to offset N and read N bytes operations succeed immediately if "currentOffset + N <= bundleLength" and fail otherwise.

5. Return the result of running Section 3.3 with "stream" as input.

3.4. Load a response from a bundle

The result of Load a bundle's metadata maps each URL and Variant-Key ([I-D.ietf-httpbis-variants]) to a response, which consists of headers and a payload. The headers can be loaded from the bundle's stream before waiting for the payload, and similarly the payload can be streamed to downstream consumers.

response = [headers: bstr .cbor headers, payload: bstr]

To implement Section 2.3, the parser MUST run the following steps, taking the bundle's "stream", a "request" ([FETCH]), and a "responseMetadata" returned by Section 2.2 .

1. Seek to offset "responseMetadata.offset" in "stream". If this fails, return an error.

2. Read 1 byte from "stream". If this is an error or isn't "0x82", return an error.

3. Let "headerLength" be the result of getting the length of a CBOR bytestring header from "stream" (Section 3.5.2). If "headerLength" is an error, return that error.

4. If "headerLength" is 524288 (512*1024) or greater, return an error.

5. Let "headerCbor" be the result of reading "headerLength" bytes from "stream" and parsing a CBOR item from them matching the "headers" CDDL rule. If either the read or parse returns an error, return that error.

6. Let ("headers", "pseudos") be the result of converting "headerCbor" to a header list and pseudoheaders using the algorithm in Section 3.6. If this returns an error, return that error.

7. If "pseudos" does not have a key named ':status' or its size isn't 1, return an error.

8. If "pseudos[':status']" isn't exactly 3 ASCII decimal digits, return an error.

9. Let "payloadLength" be the result of getting the length of a CBOR bytestring header from "stream" (Section 3.5.2). If "payloadLength" is an error, return that error.

10. If "payloadLength" is greater than 0 and "headers" does not contain a "Content-Type" header, return an error.

    The client MUST interpret the following payload as this specified media type instead of trying to sniff a media type from the bytes of the payload, for example by appending an artificial "X-Content-Type-Options: nosniff" header field ([FETCH]) to "headers".

11. If "stream.currentOffset + payloadLength != responseMetadata.offset + responseMetadata.length", return an error.

12. Let "body" be a new body ([FETCH]) whose stream is a tee'd copy of "stream" starting at the current offset and ending after "payloadLength" bytes.

    TODO: Add the rest of the details of creating a "ReadableStream" object.

13. Let "response" be a new response ([FETCH]) whose:

    * Url list is "request"'s url list,

    * status is "pseudos[':status']",

    * header list is "headers", and

    * body is "body".

14. Return "response".

3.5. Parsing CBOR items

Parsing a bundle involves parsing many CBOR items. All of these items need to be deterministically encoded.

3.5.1. Parse a known-length item

To parse a CBOR item ([CBORbis]), optionally matching a CDDL rule ([CDDL]), from a sequence of bytes, "bytes", the parser MUST do the following:

1. If "bytes" are not a well-formed CBOR item, return an error.

2. If "bytes" does not satisfy the core deterministic encoding requirements from Section 4.2.1 of [CBORbis], return an error. This format does not use floating point values or tags, so this specification does not add any deterministic encoding rules for them.

3. If "bytes" includes extra bytes after the encoding of a CBOR item, return an error.

4. Let "item" be the result of decoding "bytes" as a CBOR item.

5. If a CDDL rule was specified, but "item" does not match it, return an error.

6. Return "item".

3.5.2. Parsing variable-length data from a bytestring

Bundles encode variable-length data in CBOR bytestrings, which are prefixed with their length. This algorithm returns the number of bytes in the variable-length item and sets the stream's current offset to the first byte of the contents.

To get the length of a CBOR bytestring header from a bundle's stream, the parser MUST do the following:

1. Let ("type", "argument") be the result of parsing the type and argument of a CBOR item from the stream (Section 3.5.3). If this returns an error, return that error.

2. If "type" is not "2", the item is not a bytestring. Return an error.

3. Return "argument".

3.5.3. Parsing the type and argument of a CBOR item

To parse the type and argument of a CBOR item from a bundle's stream, the parser MUST do the following. This algorithm returns a pair of the CBOR major type 0-7 inclusive, and a 64-bit integral argument for the CBOR item:

1. Let "firstByte" be the result of reading 1 byte from the stream. If "firstByte" is an error, return that error.

2. Let "type" be "(firstByte & 0xE0) / 0x20".

3. If "firstByte & 0x1F" is:

   0..23, inclusive Return ("type", "firstByte & 0x1F").

   24 Let "content" be the result of reading 1 byte from the stream. If "content" is an error or is less than 24, return an error.

   25 Let "content" be the result of reading 2 bytes from the stream. If "content" is an error or its first byte is 0, return an error.

   26 Let "content" be the result of reading 4 bytes from the stream. If "content" is an error or its first two bytes are 0, return an error.

   27 Let "content" be the result of reading 8 bytes from the stream. If "content" is an error or its first four bytes are 0, return an error.

   28..31, inclusive Return an error. Note: This intentionally does not support indefinite-length items.

4. Let "argument" be the big-endian integer encoded in "content".

5. Return ("type", "argument").

3.6. Interpreting CBOR HTTP headers

Bundles represent HTTP requests and responses as a list of headers, matching the following CDDL ([CDDL]):

```
headers = {* bstr => bstr}
```

Pseudo-headers starting with a ":" provide the non-header information needed to create a request or response as appropriate

To convert a CBOR item "item" into a [FETCH] header list and pseudoheaders, parsers MUST do the following:

1. If "item" doesn't match the "headers" rule in the above CDDL, return an error.

2. Let "headers" be a new header list ([FETCH]).

3. Let "pseudos" be an empty map ([INFRA]).

4. For each pair ("name", "value") in "item":

    1. If "name" contains any upper-case or non-ASCII characters, return an error. This matches the requirement in Section 8.1.2 of [RFC7540].

    2. If "name" starts with a ':':

        1. Assert: "pseudos[name]" does not exist, because CBOR maps cannot contain duplicate keys.

        2. Set "pseudos[name]" to "value".

        3. Continue.

    3. If "name" or "value" doesn't satisfy the requirements for a header in [FETCH], return an error.

    4. Assert: "headers" does not contain ([FETCH]) "name", because CBOR maps cannot contain duplicate keys and an earlier step rejected upper-case bytes.

        Note: This means that a response cannot set more than one cookie, because the "Set-Cookie" header ([RFC6265]) has to appear multiple times to set multiple cookies.

    5. Append ("name", "value") to "headers".

5. Return ("headers", "pseudos").

4. Guidelines for bundle authors

Bundles SHOULD consist of a single CBOR item satisfying the core deterministic encoding requirements (Section 3.5) and matching the "webbundle" CDDL rule in Section 3.1.

5. Security Considerations

5.1. Version skew

Bundles currently have no mechanism for ensuring that the signed exchanges they contain constitute a consistent version of those resources. Even if a website never has a security vulnerability when resources are fetched at a single time, an attacker might be able to combine a set of resources pulled from different versions of the website to build a vulnerable site. While the vulnerable site could have occurred by chance on a client's machine due to normal HTTP caching, bundling allows an attacker to guarantee that it happens. Future work in this specification might allow a bundle to constrain its resources to come from a consistent version.

5.2. Content sniffing

While modern browsers tend to trust the "Content-Type" header sent with a resource, especially when accompanied by "X-Content-Type-Options: nosniff", plugins will sometimes search for executable content buried inside a resource and execute it in the context of the origin that served the resource, leading to XSS vulnerabilities. For example, some PDF reader plugins look for "%PDF" anywhere in the first 1kB and execute the code that follows it.

The "application/webbundle" format defined above includes URLs and request headers early in the format, which an attacker could use to cause these plugins to sniff a bad content type.

To avoid vulnerabilities, in addition to the response header requirements in Section 3.2, servers are advised to only serve an "application/webbundle" resource from a domain if it would also be safe for that domain to serve the bundle's content directly, and to follow at least one of the following strategies:

1. Only serve bundles from dedicated domains that don't have access to sensitive cookies or user storage.

2. Generate bundles "offline", that is, in response to a trusted author submitting content or existing signatures reaching a certain age, rather than in response to untrusted-reader queries.

3. Do all of:

    1. If the bundle's contained URLs (e.g. in the manifest and index) are derived from the request for the bundle, percent-encode [3] ([URL]) any bytes that are greater than 0x7E or are not URL code points [4] ([URL]) in these URLs. It is

particularly important to make sure no unescaped nulls (0x00) or angle brackets (0x3C and 0x3E) appear.

2. Similarly, if the request headers for any contained resource are based on the headers sent while requesting the bundle, only include request header field names *and values* that appear in a static allowlist. Keep the set of allowed request header fields smaller than 24 elements to prevent attackers from controlling a whole CBOR length byte.

3. Restrict the number of items a request can direct the server to include in a bundle to less than 12, again to prevent attackers from controlling a whole CBOR length byte.

4. Do not reflect request header fields into the set of response headers.

If the server serves responses that are written by a potential attacker but then escaped, the "application/webbundle" format allows the attacker to use the length of the response to control a few bytes before the start of the response. Any existing mechanisms that prevent polyglot documents probably keep working in the face of this new attack, but we don't have a guarantee of that.

To encourage servers to include the "X-Content-Type-Options: nosniff" header field, clients SHOULD reject bundles served without it.

6. IANA considerations

6.1. Internet Media Type Registration

IANA is requested to register the MIME media type ([IANA.media-types]) for bundled exchanges, application/webbundle, as follows:

- Type name: application
- Subtype name: webbundle
- Required parameters:
  - v: A string denoting the version of the file format. ([RFC5234] ABNF: "version = 1*(DIGIT/%x61-7A)") The version defined in this specification is "1".

    Note: RFC EDITOR PLEASE DELETE THIS NOTE; Implementations of drafts of this specification MUST NOT use simple integers to describe their versions, and MUST instead define

implementation-specific strings to identify which draft is implemented.

- Optional parameters: N/A
- Encoding considerations: binary
- Security considerations: See Section 5 of this document.
- Interoperability considerations: N/A
- Published specification: This document
- Applications that use this media type: None yet, but it is expected that web browsers will use this format.
- Fragment identifier considerations: N/A
- Additional information:
  - Deprecated alias names for this type: N/A
  - Magic number(s): 86 48 F0 9F 8C 90 F0 9F 93 A6
  - File extension(s): .wbn
  - Macintosh file type code(s): N/A
- Person & email address to contact for further information: See the Author's Address section of this specification.
- Intended usage: COMMON
- Restrictions on usage: N/A
- Author: See the Author's Address section of this specification.
- Change controller: The IESG iesg@ietf.org [5]
- Provisional registration? Yes.

6.2. Web Bundle Section Name Registry

IANA is directed to create a new registry with the following attributes:

Name: Web Bundle Section Names

Review Process: Specification Required

Initial Assignments:

Section Name	Specification	Metadata	Metadata Fields
"index"	Section 3.3.1	Yes	"requests"
"manifest"	Section 3.3.2	Yes	"manifest"
"signatures"	Section 3.3.3	Yes	"authorities", "vouched-subsets"
"critical"	Section 3.3.4	Yes	
"responses"	Section 3.3.5	No	

Requirements on new assignments:

Section Names MUST be encoded in UTF-8.

Assignments must specify whether the section is parsed during Load a bundle's metadata (Metadata=Yes) or not (Metadata=No).

The section's specification can use the bytes making up the section, the bundle's stream (Section 2.1), and the "sectionOffsets" map (Section 3.3), as input, and MUST say if an error is returned, and otherwise what items, if any, are added to the map that Section 3.3 returns. A section's specification MAY say that, if it is present, another section is not processed.

7. References

7.1. Normative References

[appmanifest] Caceres, M., Christiansen, K., Lamouri, M., Kostiainen, A., Dolin, R., and M. Giuca, "Web App Manifest", World Wide Web Consortium WD WD-appmanifest-20180523, May 2018, <https://www.w3.org/TR/2018/WD-appmanifest-20180523>.

[CBORbis] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", draft-ietf-cbor-7049bis-07 (work in progress), August 2019.

[CDDL] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <https://www.rfc-editor.org/info/rfc8610>.

[FETCH] WHATWG, "Fetch", September 2019, <https://fetch.spec.whatwg.org/>.

[I-D.ietf-httpbis-variants] Nottingham, M., "HTTP Representation Variants", draft-ietf-httpbis-variants-05 (work in progress), March 2019.

[I-D.yasskin-http-origin-signed-responses] Yasskin, J., "Signed HTTP Exchanges", draft-yasskin-http-origin-signed-responses-06 (work in progress), July 2019.

[IANA.media-types] IANA, "Media Types", <http://www.iana.org/assignments/media-types>.

[INFRA] WHATWG, "Infra", September 2019, <https://infra.spec.whatwg.org/>.

[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>.

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

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

[URL] WHATWG, "URL", September 2019, <https://url.spec.whatwg.org/>.

7.2. Informative References

[I-D.yasskin-webpackage-use-cases]
Yasskin, J., "Use Cases and Requirements for Web Packages", draft-yasskin-webpackage-use-cases-01 (work in progress), March 2018.

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

[TLS1.3] 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>.

7.3. URIs

[1] https://www.ietf.org/mailman/listinfo/wpack

[2] https://github.com/WICG/webpackage

[3] https://url.spec.whatwg.org/#percent-encode

[4] https://url.spec.whatwg.org/#url-code-points

[5] mailto:iesg@ietf.org

Appendix A. Change Log

RFC EDITOR PLEASE DELETE THIS SECTION.

draft-02

- Fix the initial bytes of the format.
- Allow empty responses to omit their content type.
- Provisionally register application/webbundle.

draft-01

- Include only section lengths in the section index, requiring sections to be listed in order.
- Have the "index" section map URLs to sets of responses negotiated using the Variants system ([I-D.ietf-httpbis-variants]).
- Require the "manifest" to be embedded into the bundle.

- Add a content sniffing security consideration.
- Add a version string to the format and its mime type.
- Add a fallback URL in a fixed location in the format, and use that fallback URL as the primary URL of the bundle.
- Add a "signatures" section to let authorities (like domain-trusted X.509 certificates) vouch for subsets of a bundle.
- Use the CBORbis "deterministic encoding" requirements instead of "canonicalization" requirements.

Appendix B. Acknowledgements

Thanks to the Chrome loading team, especially Kinuko Yasuda and Kouhei Ueno for making the format work well when streamed.

Author's Address

Jeffrey Yasskin
Google

Email: jyasskin@chromium.org