httpbis
Internet-Draft
Intended status: Standards Track
Expires: 19 April 2026

E. Nygren
M. Bishop
Akamai Technologies

16 October 2025

HTTP/1.1 Request Smuggling Defense using Cryptographic Message Binding
draft-nygren-httpbis-http11-request-binding-00

Abstract

HTTP/1.1 Message Binding adds new hop-by-hop header fields that are cryptographically bound to requests and responses. The keys used are negotiated out-of-band from the HTTP datastream (such as via TLS Exporters). These header fields allow endpoints to detect and mitigate desynchronization attacks, such as HTTP Request Smuggling, that exist due to datastream handling differences.

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 19 April 2026.

Copyright Notice

Table of Contents

1. Introduction 2
1.1. Motivation 2
1.2. Mitigation Overview 3
1.3. Illustrative Example 4
2. Conventions and Definitions 5
3. Bound Message Header Protocol 5
3.1. Request/Response Serials 6
3.2. Binding Key 6
3.3. Header Specification 6
3.4. For Discussion: Additional Attributes to Bind? 7
3.5. Intermediary Request Handling 8
3.6. Downstream Server Request Handling 8
3.7. Intermediary Response Handling 9
3.8. Handling 100 Continue and 103 Early Hints 10
3.9. Retrying Requests 10
3.10. Handling TLS 1.3 Early Data 10
4. Use with HTTPS over TLS 10
4.1. Negotiation 10
4.2. Key Derivation using TLS Exporters 11
5. Security Considerations 11
5.1. Handling detection of desynchronized connections 12
5.2. Logging failures 12
5.3. Use of keys negotiated out-of-band 12
6. Privacy considerations 13
7. IANA Considerations 13
8. References 13
8.1. Normative References 13
8.2. Informative References 14
Appendix A. Appendix: Alternate Approaches and Similar Protocols 15
Appendix B. Appendix: Bikeshed Topics 15
Acknowledgments 15
Authors' Addresses 15

1. Introduction

1.1. Motivation

HTTP Request Smuggling is a class of desynchronization attack [HTTPSYNC] where a malicious endpoint can cause a chain of other endpoints to get confused about HTTP request framing due to attributes of the HTTP/1.1 protocol leading to ambiguities in interpretation and variations in implementation. For example, if in a flow of:

User Agent => Intermediary => Origin Server

the User Agent can send an HTTP request header field with two Content-Length header fields and a Body that contains a second smuggled HTTP request after one of the content lengths. If the Intermediary and Origin Server interpret the request in different ways, the Intermediary might think that there was one request while the Origin Server thinks there are now two requests. Not only would the first request get smuggled past Intermediary defenses, if there is a second real request (so a total of three requests if you include the smuggled one) then the Intermediary might cache the contents of the smuggled response with the cache key of the third request.

There are nigh-infinite variations on this class of attack against HTTP/1.1 with frequent vulnerabilities being found and fixed. While some of these are implemenation bugs, others are due to underspecification in the HTTP/1.1 protocol itself. This latter case is hard for any single party to fix, hence where this specification can act as an additonal line of defense.

While HTTP/2 and HTTP/3 are better ([RFC9113] [RFC9114]), conversions between HTTP versions can also be vectors for vulnerabilities here to creep in. Additionally, a malicious User Agent could force an HTTP/1.1 connection to pollute shared resources (a cache or persistent connection) shared with other User Agents using newer HTTP protocols. Furthermore, the simplicity of HTTP/1.1 and large legacy code bases mean that there is extensive use of HTTP/1.1 in Intermediaries such as reverse proxies in the ecosystem: Origin Servers themselves may have an implementation where an Intermediary proxy fronts application servers, each of which having distinct HTTP implementations potentially from different vendors.

1.2. Mitigation Overview

The key concept of this specification is for HTTP/1.1 endpoints (such as an Intermediary and an Origin Server) to be able to share information about their state (e.g., which request/response they think they’re parsing) in a way that is cryptographically bound to the hop-by-hop TLS connection. Since the attacker has no access to the key used for the cryptographic binding, this allows the endpoints to detect desynchronization and fail out but without needing changes to the HTTP/1.1 protocol itself. This shared key is then used to authenticate newly introduced hop-by-hop header fields, binding information in those header fields (which includes sequential request/response serial numbers) to the request. Cases where requests or responses do become desynchronized will be detected due to invalid bound header fields (either due to failing to validate or not matching what is expected).

While "Request Framing Confusion" attacks (such as HTTP Request Smuggling or HRS) are one of the most common forms of HTTP Processing Discrepancy attacks, other types of attacks such as Host Confusion can also cause problems ([HTTPSYNC]). This specification focuses on the former, but as it evolves we may be able to extend the approach taken to defend against other forms of attacks such as Host Confusion and Path Confusion, as well as to protect header fields added by Intermediaries.

(FOR DISCUSSION: How broadly do we want to scope this specification? How much do we include here, and how much do we leave hooks to enable future extension? At the moment this is intentionally in a middle-ground, and we may either want to simplify or make more general.)

HTTP endpoints communicating HTTPS over TLS use TLS Exporters to obtain the key used for the binding ([RFC8446], Section 7.5 [RFC5705]), enabling both endpoints of a connection to securely derive this key out-of-band from the request flow in a way that can’t be tampered with. The use of Message Binding header fields is also negotiated during the TLS handshake.

The key used for the binding is abstracted out, so proprietary implementations not using TLS can distribute the key in some other manner, such as in a preface attribute that could be added to the PROXY protocol [PROXY].

1.3. Illustrative Example

In an example HRS attack from a malicious User Agent to an Origin Server through an Intermediary, the request might start out normally but the malicious User Agent smuggles a second malicious request into the initial request (e.g., due to a bug in the Intermediary or due to the Intermediary and Origin Server interpreting the HTTP/1.1 protocol slightly differently).

(TODO: Add a diagram)

The net result is that the Intermediary and Origin Server get desynchronized as to how requests and responses line up. When the malicious User Agent makes a second request, it gets back the response to the smuggled request, and a caching Intermediary may actually cache the response to the smuggled request with the cache key of this second request. This means the attacker can not only bypass any controls the Intermediary may be implementing, but may also be able to poison its cache.

With the proposed mitigation, the Intermediary augments the first and second requests (from its perspective) with cryptographically protected hop-by-hop Bound-Request header fields indicating a serial number (e.g., 1 and 2). While the Origin Server is able to validate the header field in the first request, the smuggled request is missing the header field (and even if the attacker tried to add one it would fail validation due to the attacker not having the cryptographic secret). This allows the Origin Server to detect the desynchronization, enabling it to refuse to process the smuggled request and terminate the connection.

(TODO: Add a diagram)

Request Smuggling is a family of attacks with many variations. This is why it’s valuable to include the request and response binding hop-by-hop header fields in both directions, as in some other variations it might be possible for things to get reordered such that an Intermediary making request A with serial=1 might get back a response for a request C with serial=2 and needs to be able to fail on that as well, as well as any wide range of other similar cases of desynchronization.

The need for a cryptographic binding to the channel between the Intermediary and Downstream Server (e.g., with TLS Exporters) is required to prevent the malicious User Agent from including a fake request binding header field in what is being smuggled in (which by its nature may be invisible to the Intermediary due to some bug or vulnerability).

2. Conventions and Definitions

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

3. Bound Message Header Protocol

This specification introduces new hop-by-hop Bound-Request and Bound-Response message header fields, which use [RFC8941] structured fields. These header fields convey a request/response Serial number, additional attributes, and a cryptographic binding.

As these are hop-by-hop header fields they are added by the endpoints on the HTTP/1.1 persistent connection ([RFC9112]). Below we refer to the outbound endpoint making the request as the Intermediary ([RFC9110], Section 3.7) and the inbound endpoint receiving the

request and issuing a response as the Downstream Server. The Downstream Server may be either another Intermediary or an Origin Server.

Intermediaries and Downstream Servers MUST NOT exchange Bound-Request and Bound-Response header fields unless they have mutually negotiated this protocol, either as described below in Section 4.1 or via some other out-of-band mechanism. If the User Agent and Downstream Server have negotiated using this protocol for a connection, they MUST send a Bound-Request and Bound-Response header fields in all requests and responses on that connection.

3.1. Request/Response Serials

The Request Serial ($req_serial) is a counter starting at 1 for the initial request in an HTTP/1.1 persistent connection, and then incrementing by 1 for each subsequent request. The Response Serial ($resp_serial) for a response is then reflected back to match the Request Serial from the corresponding request.

3.2. Binding Key

The Binding Key is a binary cryptographic value that is associated with the connection. Below we will refer to the binding key for requests as $req_key and the binding key for responses as $resp_key.

With HTTPS over TLS the binding keys MUST be derived as described in Section 4.2.

3.3. Header Specification

The Bound-Request and Bound-Response header fields are specified as an integer item (the Serial) followed by a parameter list of items.

The ABNF is as follows:

```
bound_header      = bound_header_name ":" serial ";" OWS
                    "method=" method ";" OWS
                    "authority=" authority ";" OWS
                    ("response-code" = response_code ";" OWS)?
                    "binding=" binding_value
bound_header_name = "Bound-Request" | "Bound-Response"
serial            = sf-integer
method            = sf-string
authority         = sf-string
response_code     = sf-integer
binding_value     = sf-binary
```

(TODO: restructure the ABNF to allow the parameter orders to vary)

The binding value for a request or response with a given key ($req_key or $resp_key) is constructed as:

```
 binding_value = HMAC-SHA256($key, $serial "|" $method "|" authority)
```

In the above:

* $key is the $req_key or $resp_key

* $serial is the request or response serial as a string

* $method is the HTTP request method associated with the request

* $authority is the authority ((as defined in [RFC9110], Section 4.2.3) from the normalized URI (as defined in [RFC9110], Section 4.2.3) for the request and MUST match the value in the request's Host header field

* $response_code is the response code for the response

* The binding value construct uses HMAC-SHA256 ([RFC2104])

(TODO: consider whether/how to add crypto agility for other keyed MACs)

For example, the header field added to the first request on a connection might be:

```
Bound-Request: 1; method=POST; authority=www.example.com;
               binding=:yYwktnfv9Ehgr+pvSTu67FuxxMuyCcb8K7tH9m/yrTE=:
```

3.4. For Discussion: Additional Attributes to Bind?

FOR DISCUSSION: Do we want to add in additional information to defend against additional sorts of attacks? Would we want to change how we encode these?

The reasons to include attributes into the Message Binding are:

* Information from an Intermediary or Origin Server endpoint is intermixed in the bytestream with information that was sourced from a potentially malicious User Agent. HTTP/2 and HTTP/3 use distinct pseudoheaders to encode some of these separately, but other header fields such as Client-Cert ([RFC9440]) have no such protections.

* The encoding within HTTP/1.1 is underspecified in ways that lead to ambiguity, such as with variations in Path and Host header field parsing.

Some options might include:

* Adding the :path as a parameter (or adding an attribute indicating that it should be considered included) and also binding it in.

* Having a way to more generally encode HTTP/2 pseudoheader field values in a way that is less ambiguous (converted to sf-binary?) and gets bound in.

* Including a list of header fields to bind in, and then use [RFC9421] HTTP Message Signatures or similar to protect them. This would be particularly useful for protecting header fields such as Client-Cert.

Adding more in does add more complexity and has more risks of compatibility issues. It may also be worth considering going the other direction and removing method and authority parameters.

3.5. Intermediary Request Handling

Intermediaries which have negotiated this protocol MUST add a Bound-Request header field with each request they make. The $req_serial MUST start at 1 for the first request on a persistent connection, and MUST be incremented by 1 for each subsequent request.

If the Intermediary is an Intermediary, regardless of whether or not this protocol was negotiated for the connection, it MUST remove any Bound-Request header fields that it received (prior to adding its own, if applicable).

3.6. Downstream Server Request Handling

Downstream Servers which have negotiated this protocol MUST validate the presence and contents of the Bound-Request header field prior to processing a request. Any failures MUST be detected early in request processing (such as during request parsing), and Downstream Servers MUST immediately terminate the connection without returning an error response.

Validation checks MUST include:

* Confirmation that the Bound-Request header field is present

* Confirmation that the cryptographic binding hash matches what was expected

* Confirmation that the $req_serial matches what was expected, starting at 1 for the first request on the connection and incrementing by 1 for each subsequent request

* Confirmation that the authority and method match those in the request

If the Downstream Server is an Intermediary, it MUST remove the Bound-Request header field before constructing a request to the next-hop, regardless of whether this protocol is used for the next-hop.

When constructing a response to the HTTP request the Downstream Server MUST add a Bound-Response header field with a $resp_serial matching the $req_serial of the incoming request.

If the Downstream Server is an Intermediary, it MUST first remove any Bound-Response header fields that it received, regardless of whether this protocol is used on the previous-hop.

3.7. Intermediary Response Handling

Intermediaries which have negotiated this protocol MUST validate the presence and contents of the Bound-Response header field prior to processing a response. Any failures MUST be detected early in response processing (such as during response parsing), and Intermediaries MUST immediately terminate the connection without processing any data from the response.

Validation checks MUST include:

* Confirmation that the Bound-Response header field is present

* Confirmation that the cryptographic binding MAC matches what was expected

* Confirmation that the $resp_serial matches the $req_serial of the request that the response is in-response to.

* Confirmation that the authority and method match those from the corresponding request

* Confirmation that the $response_code matches that from the response (or interim response, as discussed in Section 3.8)

The Intermediary MUST remove the Bound-Response header field before constructing a response to the previous-hop, regardless of whether this protocol is used for the previous-hop.

3.8. Handling 100 Continue and 103 Early Hints

When using 100 Continue and 103 Early Hints, the $req_serial and $resp_serial MUST remain the same and match for all interim and final responses. Each interim response MUST contain a Bound-Response header field with a response-code parameter matching the response code of the interim response.

(TODO can we safely extend this requirement to all 1xx status codes?)

3.9. Retrying Requests

Requests which are retried MUST be treated no differently than other forms of request, with their $req_serial coming from the order of the request in a persistent connection. If a request is retried over a different connection a new Bound-Request header field MUST be reconstructed corresponding to the new connection.

3.10. Handling TLS 1.3 Early Data

_TODO: define how this works with TLS 1.3 0RTT as it adds additional wrinkles. While this maybe could be made to work there (e.g., using the separate early exporter secret and a distinct space for request_serials) [RFC8446], we need to ensure that we properly handle situations where an HTTP request spans 0-RTT and 1-RTT data._

4. Use with HTTPS over TLS

4.1. Negotiation

Since the Bound-Request header field is hop-by-hop header field it is not safe to send unless the Intermediary knows that recipient supports it, will process it, and then will remove it. Intermediaries and Downstream Servers MUST NOT send Bound-Request or Bound-Response header fields on connections where they have not negotiated this protocol.

Negotiation needs to happen out-of-band (e.g., at the TLS layer) due to the nature of the attacks this is trying to mitigate.

Options for negotiation include:

* ALPS (stalled/expired) [I-D.vvv-tls-alps]

* TLS Extension Flags (waiting on implementation) [I-D.ietf-tls-tlsflags]

* An all-new TLS extension specific to this purpose, which could also make it easier to version this protocol.

Note that the first two options only support TLS 1.3 [RFC8446] between the Intermediary and Downstream Server.

It would also be preferable for the mechanism here to negotiate the supported versions of this protocol, such as if cryptographic agility or additional functionality is needed.

Application Protocols (ALPN values, per [RFC7301]) other than "http/1.1" are not supported, and a Downstream Server MUST NOT negotiate this Request-Binding protocol when negotiating an application protocol other than "http/1.1".

4.2. Key Derivation using TLS Exporters

The $req_key and $resp_key are derived using TLS Exporters.

* For TLS 1.3 this is specified in [RFC8446], Section 7.5

* For TLS 1.2 this is specified in [RFC5705]

Endpoints MAY support TLS 1.2 using [RFC5705], but if they do they MUST only use this extension when the extended master secret ([RFC7627]) extension is also used. Endpoints MUST NOT use this protocol for versions of TLS prior to 1.2.

The request and response keys are constructed for a connection with:

```
$req_key = TLS-Exporter("HTTP-Request-Binding", "request-"+$alpn, 32)
$resp_key = TLS-Exporter("HTTP-Request-Binding", "response-"+$alpn, 32)
```

The added context ensures that we get different keys derived for different negotiated ALPNs. When HTTP/1.1 was negotiated without an ALPN, $alpn SHALL be http/1.1.

When this extension is negotiated, HTTP requests that indicate an HTTP-version other than HTTP/1.1 MUST be rejected, with the connection closed prior to sending an HTTP-layer response.

5. Security Considerations

5.1. Handling detection of desynchronized connections

When an endpoint detects desynchronization (due to a missing or invalid Message Binding header field) it needs to consider itself to be in an unknown, inconsistent, and potentially adversary-controlled state. Any processing that happens past this point for this or other requests on the connection is dangerous and suspect, as nothing in the connection bytestream can be trusted at this point. Letting the request or response get past validation failures during parsing would leave the endpoint vulnerable and might execute smuggled instructions.

Returning an HTTP error response would be bad as this response would be desynchronized and could be cached. While breaking the connection does not provide information to Intermediaries as to why things broke, it is imperative to terminate immediately.

TODO: explore if there may be a way to use a TLS alert to signal that badness happened to the other endpoint.

5.2. Logging failures

Endpoints SHOULD log information indicating why the request or connection failed. Even more care than usual needs to be taken handling information received as there is no way to distinguish information as having come from a potentially trusted Intermediary vs having come from a malicious adversary.

Downstream Servers logging information from detected smuggled requests need to take care as all information is suspect. It is critical that validation (and fail-out) happens very early in handling the request, such as during the request/response parsing itself. Even logging things from the smuggled request must be handled very carefully.

5.3. Use of keys negotiated out-of-band

With the use of TLS Exporters each connection gets a unique pair of $req_key and $resp_key. If an alternate mechanism is used by proprietary implementations to exchange these keys then they MUST be unique per connection. Otherwise an attacker who can get a request header reflected back from one connection might be able to replay it in another connection.

6. Privacy considerations

Due to this protocol primarily being used between Intermediaries and Downstream Servers, information sent by the Intermediary during the TLS handshake for negotiation does not cause privacy issues for end-users. If this protocol were to be extended into end-user User Agents as well, more evaluation of privacy considerations would be warranted.

7. IANA Considerations

TODO: Add IANA considerations for the HTTP Headers, for TLS Exporter labels, and for the TLS extension details used for negotiation.

8. References

8.1. Normative References

[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/rfc/rfc2104>.

[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/rfc/rfc2119>.

[RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, DOI 10.17487/RFC5705, March 2010, <https://www.rfc-editor.org/rfc/rfc5705>.

[RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <https://www.rfc-editor.org/rfc/rfc7301>.

[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, <https://www.rfc-editor.org/rfc/rfc7627>.

[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/rfc/rfc8174>.

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

[RFC8941] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/rfc/rfc8941>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[RFC9112] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1", STD 99, RFC 9112, DOI 10.17487/RFC9112, June 2022, <https://www.rfc-editor.org/rfc/rfc9112>.

8.2. Informative References

[HTTPSYNC] Topcuoglu, C., Onarlioglu, K., Sprecher, S., Kirda, E., and Northeastern University, "HTTP Request Synchronization Defeats Discrepancy Attacks", arXiv 2510.09952, October 2025, <https://arxiv.org/abs/2510.09952>.

[I-D.ietf-tls-tlsflags] Nir, Y., "A Flags Extension for TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-tlsflags-16, 14 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-tlsflags-16>.

[I-D.vvv-tls-alps] Benjamin, D. and V. Vasiliev, "TLS Application-Layer Protocol Settings Extension", Work in Progress, Internet-Draft, draft-vvv-tls-alps-01, 21 September 2020, <https://datatracker.ietf.org/doc/html/draft-vvv-tls-alps-01>.

[PROXY] HAProxy Technologies, "The PROXY protocol", 2017, <https://www.haproxy.org/download/1.8/doc/proxy-protocol.txt>.

[RFC9113] Thomson, M., Ed. and C. Benfield, Ed., "HTTP/2", RFC 9113, DOI 10.17487/RFC9113, June 2022, <https://www.rfc-editor.org/rfc/rfc9113>.

[RFC9114] Bishop, M., Ed., "HTTP/3", RFC 9114, DOI 10.17487/RFC9114, June 2022, <https://www.rfc-editor.org/rfc/rfc9114>.

[RFC9261] Sullivan, N., "Exported Authenticators in TLS", RFC 9261, DOI 10.17487/RFC9261, July 2022, <https://www.rfc-editor.org/rfc/rfc9261>.

[RFC9421] Backman, A., Ed., Richer, J., Ed., and M. Sporny, "HTTP Message Signatures", RFC 9421, DOI 10.17487/RFC9421, February 2024, <https://www.rfc-editor.org/rfc/rfc9421>.

[RFC9440] Campbell, B. and M. Bishop, "Client-Cert HTTP Header Field", RFC 9440, DOI 10.17487/RFC9440, July 2023, <https://www.rfc-editor.org/rfc/rfc9440>.

Appendix A. Appendix: Alternate Approaches and Similar Protocols

TLS Exporters are used in other protocols such as [RFC9261] (Exported Authenticators in TLS). While it is meant as a building block, it requires round-trips for some scenarios which would make it not suitable here.

Appendix B. Appendix: Bikeshed Topics

Some details to work through include:

* Should the starting serial be 1 or 0?

Acknowledgments

The authors would like to thank Kaan Onarlioglu, Rich Salz, Benjamin Kaduk, Uttaran Dutta, and others who have contributed to this proposal.

Authors' Addresses

Erik Nygren
Akamai Technologies
Email: erik+ietf@nygren.org

Mike Bishop
Akamai Technologies
Email: mbishop@evequefou.be