Stream: Internet Engineering Task Force (IETF)
RFC: 9931
Updates: 9112, 9298
Category: Standards Track
Published: March 2026
ISSN: 2070-1721
Author: B. Schwartz
Meta Platforms, Inc.

RFC 9931
Security Considerations for Optimistic Protocol Transitions in HTTP/1.1

Abstract

In HTTP/1.1, the client can request a change to a new protocol on the existing connection. This document discusses the security considerations that apply to data sent by the client before this request is confirmed and adds new requirements to RFCs 9112 and 9298 to avoid related security issues.

Status of This Memo

Copyright Notice

Table of Contents

1. Overview

This document discusses certain security considerations that arise when switching from HTTP/1.1 to a different protocol on the same connection. It provides:

- a review of the relevant standards,
- a discussion of the security risks that may apply if a client sends data before the transition is confirmed,
- a security evaluation of existing upgrade tokens, and
- guidance for implementations and future standards documents.

Updates to [HTTP/1.1] and [CONNECT-UDP], including new normative requirements, are provided in Sections 8 and 6.3, respectively.

2. Requirements Language

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. Background

In HTTP/1.1 [HTTP/1.1] and later, a single connection can be used for many requests. In HTTP/2 [HTTP/2] and HTTP/3 [HTTP/3], these requests can be multiplexed, as each request is distinguished explicitly by its stream ID. However, in HTTP/1.1, requests are strictly sequential, and each new request is distinguished implicitly by the closure of the preceding request.

HTTP/1.1 is also the only version of HTTP that allows the client to change the protocol used for the remainder of the connection. There are two mechanisms to request such a protocol transition. One mechanism is the Upgrade header field ([HTTP], Section 7.8). When included in a request, this field indicates that the client would like to use this connection for a protocol other than HTTP/1.1. The server replies with a 101 (Switching Protocols) status code if it accepts the protocol change ([HTTP], Section 15.2.2).

The other mechanism is the HTTP CONNECT method (Section 9.3.6 of [HTTP]). This method indicates that the client wishes to establish a TCP connection to the specified host and port. If accepted, the server replies with a 2xx (Successful) response to indicate that the request was accepted and a TCP connection was established. After this point, the TCP connection is acting as a TCP tunnel, not an HTTP/1.1 connection.

Both of these mechanisms also permit the server to reject the request. For example, Section 7.8 of [HTTP] says:

> A server **MAY** ignore a received Upgrade header field if it wishes to continue using the current protocol on that connection.

and Section 9.3.6 of [HTTP] says:

> A server **MUST** reject a CONNECT request that targets an empty or invalid port number, typically by responding with a 400 (Bad Request) status code.

Rejected upgrades are common and can happen for a variety of reasons, such as:

- The server does not support any of the client's indicated upgrade tokens (i.e., the client's proposed new protocols), so it continues to use HTTP/1.1.
- The server knows that an upgrade to the offered protocol will not provide any improvement over HTTP/1.1 for this request to this resource, so it chooses to respond in HTTP/1.1.
- The server requires the client to authenticate before upgrading the protocol, so it replies with the status code 401 (Authentication Required) and provides a challenge in an Authorization response header field ([HTTP], Section 11.6.2).
- The resource has moved, so the server replies with a 3xx (Redirection) status code ([HTTP], Section 3.4).

Similarly, servers frequently reject HTTP CONNECT requests, such as when:

- The server does not support HTTP CONNECT.
- The specified destination is not allowed under server policy.
- The destination cannot be resolved, is unreachable, or does not accept the connection.
- The proxy requires the client to authenticate before proceeding.

After rejecting a request, the server will continue to interpret bytes received on that connection in accordance with HTTP/1.1.

Section 7.8 of [HTTP] also states:

> A client cannot begin using an upgraded protocol on the connection until it has completely sent the request message (i.e., the client can't change the protocol it is sending in the middle of a message).

In other words, completion of the request message is a **necessary** condition for the client to begin using the new protocol. However, it is important to clarify that this is not a **sufficient** condition because the server might reject the request.

In some cases, the client might predict that the server is likely to accept a requested protocol transition. For example, if a request using an upgrade token recently succeeded, the client might expect that subsequent requests with the same upgrade token will also succeed. If this expectation is correct, the client can often reduce delay by immediately sending the first bytes of the new protocol "optimistically", without waiting for the server's response. This document explores the security implications of this "optimistic" behavior.

4. Possible Security Issues

When there are only two distinct parties involved in an HTTP/1.1 connection (i.e., the client and the server), protocol transitions introduce no new security issues: Each party must already be prepared for the other to send arbitrary data on the connection at any time. However, HTTP connections often involve more than two parties if the requests or responses include third-party data. For example, a browser (party 1) might send an HTTP request to an origin (party 2) with the path, headers, or content controlled by a website from a different origin (party 3). Post-transition protocols, such as WebSocket [WEBSOCKET], are also frequently used to convey data chosen by a third party.

If the third-party data source is untrusted, then the data it provides is potentially "attacker-controlled". The combination of attacker-controlled data and optimistic protocol transitions results in two significant security issues.

4.1. Request Smuggling

In a Request Smuggling attack ([HTTP/1.1], Section 11.2), the attacker-controlled data is chosen in such a way that it is interpreted by the server as an additional HTTP request. These attacks allow the attacker to speak on behalf of the client while bypassing the client's own rules about what requests it will issue. Request Smuggling can occur if the client and server have distinct interpretations of the data that flows between them.

If the server accepts a protocol transition request, it interprets the subsequent bytes in accordance with the new protocol. If it rejects the request, it interprets those bytes as HTTP/1.1. However, the client cannot know which interpretation the server will take until it receives the server's response status code. If it uses the new protocol optimistically, this creates a risk that the server will interpret attacker-controlled data in the new protocol as an additional HTTP request issued by the client.

As a trivial example, consider an HTTP CONNECT client providing connectivity to an untrusted application. If the client is authenticated to the proxy server using a connection-level authentication method such as TLS Client Certificates ([TLS], Section 4.4.2), the attacker could send an HTTP/1.1 POST request ([HTTP], Section 9.3.3) for the proxy server at the beginning of its TCP connection. If the client delivers this data optimistically, and the CONNECT request fails, the server would misinterpret the application's data as a subsequent authenticated request issued by the client.

```
## REQUESTS ##

# The malicious application requests a TCP connection to a nonexistent
# destination, which will fail.
CONNECT no-such-destination.example:443 HTTP/1.1
Host: no-such-destination.example:443

# Before connection fails, the malicious application sends data on the
# proxied TCP connection that forms a valid POST request to the proxy.
# The vulnerable client optimistically forwards this data to the proxy.
POST /upload HTTP/1.1
Host: proxy.example
Content-Length: 123456

<POST body controlled by the malicious application>

## RESPONSES ##

# When TCP connection establishment fails, the proxy responds by
# rejecting the CONNECT request, but the client has already forwarded
# the malicious TCP payload data to the proxy.
HTTP/1.1 504 Gateway Timeout
Content-Length: 0

# The proxy interprets the smuggled POST request as coming from the
# client.  If connection-based authentication is in use (e.g., using
# TLS client certificate authentication), the proxy treats this
# malicious request as authenticated.
HTTP/1.1 200 OK
Content-Length: 0
```

Figure 1: Example Request Smuggling Attack Using HTTP CONNECT

4.2. Parser Exploits

A related category of attacks use protocol disagreement to exploit vulnerabilities in the server's request parsing logic. These attacks apply when the HTTP client is trusted by the server, but the post-transition data source is not. If the server software was developed under the assumption that some or all of the HTTP request data is not attacker-controlled, optimistic transmission can cause this assumption to be violated, exposing vulnerabilities in the server's HTTP request parser.

5. Operational Issues

If the server rejects the transition request, the connection can continue to be used for HTTP/1.1. There is no general requirement to close the connection in response to a rejected transition, and keeping the connection open has performance advantages if additional HTTP requests to this server are likely. Thus, it is normally inappropriate to close the connection in response to a rejected transition.

6. Impact on HTTP Upgrade with Existing Upgrade Tokens

This section describes the impact of this document's considerations on some registered upgrade tokens [IANA-UPGR] that are believed to be in use at the time of writing.

6.1. "TLS"

The "TLS" family of upgrade tokens was defined in [RFC2817], which correctly highlights the possibility of the server rejecting the upgrade. If a client ignores this possibility and sends TLS data optimistically, the result cannot be valid HTTP/1.1: The first octet of a TLS connection must be 22 (ContentType.handshake), but this is not an allowed character in an HTTP/1.1 method (see [TLS], Section 5.1 and [HTTP/1.1], Section 3). A compliant HTTP/1.1 server will treat this as a parsing error and close the connection without processing further requests.

6.2. "WebSocket"/"websocket"

Section 4.1 of [WEBSOCKET] says:

> Once the client's opening handshake has been sent, the client **MUST** wait for a response from the server before sending any further data.

Thus, optimistic use of HTTP Upgrade is already forbidden in the WebSocket protocol. Additionally, the WebSocket protocol requires high-entropy masking of client-to-server frames (Section 5.1 of [WEBSOCKET]).

6.3. "connect-udp"

Section 5 of [CONNECT-UDP] says:

> A client **MAY** optimistically start sending UDP packets in HTTP Datagrams before receiving the response to its UDP proxying request.

However, in HTTP/1.1, this "proxying request" is an HTTP Upgrade request. This upgrade is likely to be rejected in certain circumstances, such as when the UDP destination address (which is attacker-controlled) is invalid. Additionally, the contents of the "connect-udp" protocol stream can include untrusted material (i.e., the UDP packets, which might come from other applications on the client device). This creates the possibility of Request Smuggling attacks. To avoid these concerns, this document replaces that text to exclude HTTP/1.1 from any optimistic sending, as follows:

> A client **MAY** optimistically start sending UDP packets in HTTP Datagrams before receiving the response to its UDP proxying request but only if the HTTP version in use is HTTP/2 or later. Clients **MUST NOT** send UDP packets optimistically in HTTP/1.x due to the risk of Request Smuggling attacks.

6.4. "connect-ip"

The "connect-ip" upgrade token is defined in [CONNECT-IP]. Section 11 of [CONNECT-IP] forbids clients from sending packets optimistically in HTTP/1.1, avoiding this issue.

7. Guidance for Future Upgrade Tokens

There are now several good examples of designs that reduce or eliminate the security concerns discussed in this document and may be applicable in future specifications:

- Forbid optimistic use of HTTP Upgrade (Section 4.1 of [WEBSOCKET] and Section 11 of [CONNECT-IP]).
- Embed a fixed preamble that deliberately terminates HTTP/1.1 processing (Section 3.4 of [HTTP/2]).
- Apply high-entropy masking of client-to-server data (Section 5.1 of [WEBSOCKET]).

Future specifications for upgrade tokens should account for the security issues discussed here and provide clear guidance on how implementations can avoid them.

7.1. Selection of Request Methods

Some upgrade tokens, such as "TLS", are defined for use with any ordinary HTTP method. The upgraded protocol continues to provide HTTP semantics and will convey the response to this HTTP request.

The other upgrade tokens mentioned in Section 6 do not preserve HTTP semantics, so the method is not relevant. All of these upgrade tokens are specified only for GET requests with no content.

Future specifications for upgrade tokens should restrict their use to GET requests with no content if the HTTP method is otherwise irrelevant and the request does not need to carry any message content. This improves consistency with other upgrade tokens and simplifies server implementation.

8. Requirements for HTTP CONNECT

This document updates [HTTP/1.1] to include the remaining text of this section. The requirements in this section apply only to HTTP/1.1.

Proxy clients that send CONNECT requests on behalf of untrusted TCP clients **MUST** do one or both of the following:

1. Wait for a 2xx (Successful) response before forwarding any TCP payload data.
2. Send a "Connection: close" request header.

Proxy clients that don't implement at least one of these two behaviors are vulnerable to a trivial Request Smuggling attack ([HTTP/1.1], Section 11.2).

At the time of writing, some proxy clients are believed to be vulnerable as described. As a mitigation, proxy servers **MUST** close the underlying connection when rejecting a CONNECT request without processing any further requests on that connection. This requirement applies whether or not the request includes a "close" connection option.

Note that this mitigation will frequently cause slower connection establishment for correctly implemented clients, especially when returning a 407 (Proxy Authentication Required) response. This performance loss can be avoided by using HTTP/2 or HTTP/3, which are not vulnerable to this attack.

As a performance optimization, proxy servers **MAY** disable this mitigation if the client is known to wait for a 2xx (Successful) response before forwarding untrusted TCP payload data (i.e., complying with item 1 above). Proxy servers can identify compliant clients using the request's User-Agent header field and the user agent vendor's documentation regarding its compliance.

9. Security Considerations

This document describes security considerations related to optimistic use of protocol transitions in HTTP/1.1.

10. IANA Considerations

This document has no IANA actions.

11. References

11.1. Normative References

[CONNECT-UDP] Schinazi, D., "Proxying UDP in HTTP", RFC 9298, DOI 10.17487/RFC9298, August 2022, <https://www.rfc-editor.org/info/rfc9298>.

[HTTP] 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/info/rfc9110>.

[HTTP/1.1] 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/info/rfc9112>.

[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>.

[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>.

11.2. Informative References

[CONNECT-IP] Pauly, T., Ed., Schinazi, D., Chernyakhovsky, A., Kühlewind, M., and M. Westerlund, "Proxying IP in HTTP", RFC 9484, DOI 10.17487/RFC9484, October 2023, <https://www.rfc-editor.org/info/rfc9484>.

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

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

[IANA-UPGR] IANA, "Hypertext Transfer Protocol (HTTP) Upgrade Token Registry", <https://www.iana.org/assignments/http-upgrade-tokens/>.

[RFC2817] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, DOI 10.17487/RFC2817, May 2000, <https://www.rfc-editor.org/info/rfc2817>.

[TLS] 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>.

[WEBSOCKET] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <https://www.rfc-editor.org/info/rfc6455>.

Acknowledgments

This document benefited from valuable reviews and suggestions by:

- Mike Bishop
- Mohamed Boucadair
- Gorry Fairhurst
- Mark Nottingham
- Kazuho Oku
- Lucas Pardue
- David Schinazi
- Glenn Strauss
- Michael Sweet
- Willy Tarreau
- Martin Thomson

Author's Address

Benjamin M. Schwartz
Meta Platforms, Inc.
Email: ietf@bemasc.net