Workgroup: Network Working Group
Internet-Draft: draft-ietf-tls-snip-01
Published: 16 February 2022
Intended Status: Informational
Expires: 20 August 2022
Authors: M. Thomson
Mozilla

Secure Negotiation of Incompatible Protocols in TLS

Abstract

An extension is defined for TLS that allows a client and server to detect an attempt to force the use of less-preferred application protocol even where protocol options are incompatible. This supplements application-layer protocol negotiation (ALPN), which allows choices between compatible protocols to be authenticated.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

With increased diversity in protocol choice, some applications are able to use one of several semantically-equivalent protocols to achieve their goals. This is particularly notable in HTTP where there are currently three distinct protocols: HTTP/1.1 [HTTP11], HTTP/2 [HTTP2], and HTTP/3 [HTTP3]. This is also true of protocols that support variants based on both TLS [TLS] and DTLS [DTLS].

For protocols that are mutually compatible, Application-Layer Protocol Negotiation (ALPN; [ALPN]) provides a secure way to negotiate protocol selection.

In ALPN, the client offers a list of options in a TLS ClientHello and the server chooses the option that it most prefers. A downgrade attack occurs where both client and server support a protocol that the server prefers more than than the selected protocol. ALPN protects against this attack by ensuring that the server is aware of all options the client supports and including those options and the server choice under the integrity protection provided by the TLS handshake.

Downgrade protection in ALPN functions because protocol negotiation is part of the TLS handshake. The introduction of semantically-equivalent protocols that use incompatible handshakes introduces new opportunities for downgrade attack. For instance, it is not possible to negotiate the use of HTTP/2 based on an attempt to connect using

HTTP/3. The former relies on TCP, whereas the latter uses UDP. These protocols are therefore mutually incompatible and ALPN cannot be used to securely select between the two.

This document defines an extension to TLS that allows clients to discover when a server supports alternative protocols that are incompatible with the protocol in use. This might be used to detect a downgrade attack.

Downgrade protection for incompatible protocols only works for services provided by the same logical server (see Section 3.2). That is, the protection only applies to servers that operate from the same IP address and port number from the perspective of the client.

This extension is motivated by the addition of new protocols such as HTTP/3 [HTTP3] that are semantically equivalent, but incompatible with existing protocols.

These downgrade protections are intended to work for any method that a client might use to discover that a server supports a particular protocol. Special considerations for HTTP Alternative Services [ALTSVC] is included in Section 4.3 and a discussion of SVCB [SVCB] can be found in Appendix C.

2. Terminology

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.

Two protocols are considered "compatible" if it is possible to negotiate either using the same connection attempt. In comparison, protocols are "incompatible" if they require separate attempts to establish a connection.

3. Incompatible Protocol Selection

This document extends the authentication protections provided by TLS to cover negotiation of incompatible protocols.

This is complementary to ALPN [ALPN], which only protects the negotiation of compatible protocols. In ALPN, the client presents a set of compatible options and the server chooses its most preferred.

This extension works by having a server offer a list of incompatible protocols that it supports on the same logical server (see Section 3.2). How clients use this information will depend on client policy.

3.1. Client Policy

A client has to choose between incompatible options before making a connection attempt. Thefore, this document does not define a negotiation mechanism, it only provides authenticated information that a client can use.

Importantly, detecting a potential downgrade between incompatible protocols does not automatically imply that a client abandon a connection attempt. It only provides the client with authenticated information that can help with making a decision. What a client does with this information is left to client policy.

For a protocol like HTTP/3, this might not result in the client choosing to use HTTP/3, even if HTTP/3 is preferred and the server indicates that a service endpoint supporting HTTP/3 is available. Blocking of UDP or QUIC is known to be widespread. As a result, clients might adopt a policy of tolerating a downgrade to a TCP-based version of HTTP, even if HTTP/3 were preferred. However, as blocking of UDP is highly correlated by access network, clients that are able to establish HTTP/3 connections to some servers might choose to apply a stricter policy when a server that indicates HTTP/3 support is unreachable.

3.2. Logical Servers

This document relies on the notion of a logical server for determining how a client interprets information about incompatible protocols.

Clients can assume availability of incompatible protocols across the set of endpoints that share an IP version, IP address, and port number with the TLS server that provides the incompatible_protocols extension.

This definition includes a port number that is independent of the protocol that is used. Any protocol that defines a port number is considered to be equivalent. In particular, incompatible protocols can be deployed to TCP, UDP, SCTP, or DCCP ports as long as the IP address and port number is the same.

This determination is made from the perspective of a client. This means that server operators need to be aware of all instances that might answer to the same IP address and port; see Section 5.

4. Authenticating Incompatible Protocols

The incompatible_protocols(TBD) TLS extension provides clients with information about the incompatible protocols that are supported by the same logical server; see Section 3.2 for a definition of a logical server.

```
enum {
    incompatible_protocols(TBD), (65535)
} ExtensionType;
```

A client that supports the extension advertises an empty extension. In response, a server that supports this extension includes a list of application protocol identifiers. The "extension_data" field of the server extension uses the ProtocolName type defined in [ALPN]. This syntax is shown in Figure 1.

```
opaque ProtocolName<1..2^8-1>;  // From RFC 7301
ProtocolName IncompatibleProtocol;

struct {
  select (Handshake.msg_type) {
    case client_hello:
      Empty;
    case encrypted_extensions:
      IncompatibleProtocol incompatible_protocols<3..2^16-1>;
  };
} IncompatibleProtocols;
```

Figure 1: TLS Syntax for incompatible_protocols Extension

This extension only applies to the ClientHello and EncryptedExtensions messages. An implementation that receives this extension in any other handshake message MUST send a fatal illegal_parameter alert.

A client offers an empty extension to indicate that is wishes to receive information about incompatible protocols supported by the (logical) server.

A server deployment that supports multiple incompatible protocols MAY advertise all protocols that are supported by the same logical server. A server needs to ensure that protocols advertised in this fashion are available to the client.

A server MUST omit any compatible protocols from this extension. That is, any protocol that the server might be able to select, had the client offered the protocol in the application_layer_protocol_negotiation extension. In comparison, clients are expected to include all compatible protocols in the application_layer_protocol_negotiation extension.

Information presented by the server is only valid at the time it is provided. A client can act on that information immediately, but it cannot retain the information on the expectation that it will be valid later. A server therefore only needs to consider providing information that is current for a period that would allow the client to act, which might amount to a few seconds.

4.1. Validation

If a client has discovered server endpoints for a preferred protocol that point to the same logical server, receiving an incompatible_protocols extension that includes that protocol is a strong indication of a potential downgrade attack.

In response to detecting a potential downgrade attack, a client might abandon the current connection attempt and report an error.

A client might support an incompatible protocol, but chooses not to attempt its use under normal conditions might choose not to fail if it learns that the protocol is supported by the server. This client might instead make a connection attempt or initiate discovery for that protocol when it learns that it is available.

4.2. QUIC Version Negotiation

QUIC enables the definition of incompatible protocols that share a port. The incompatible_protocols extension can be used to authenticate the choice of application protocols across incompatible QUIC version. QUIC version negotiation [QUIC-VN] is used to authenticate the choice of QUIC version.

As there are two potentially competing sets of preferences at different protocol layers, clients need to set preferences for QUIC version and application protocol are consistent.

For example, if application protocol A exclusively uses QUIC version X and application protocol B exclusively uses QUIC version Y, setting a preference for both A and Y will result in one or other option not being selected. This would result in failure if the client applied a policy that regarded either downgrade as an error.

4.3. HTTP Alternative Services

It is possible to select incompatible protocols based on an established connection. The Alternative Services [ALTSVC] bootstrapping in HTTP/3 [HTTP3] is not vulnerable to downgrade as the signal is exchanged over an authenticated connection. A server can advertise the presence of an endpoint that supports HTTP/3 using an HTTP/2 or HTTP/1.1 connection.

A client can choose to ignore incompatible protocols when attempting to use an alternative service.

5. Operational Considerations

By listing incompatible protocols a server needs to be certain that the incompatible protocols are available. Ensuring that this information is correct might need some amount of coordination in server deployments. In particular, coordination is important if a load balancer distributes load for a single IP address to multiple server instances, or where anycast [BCP126] is used.

Incompatible protocols can only be listed in the incompatible_protocols extension when those protocols are deployed across all server instances. A client might regard lack of availability for an advertised protocol as a downgrade attack, which could lead to service outages for those clients.

Server deployments can choose not to provide information about incompatible protocols might avoid the operational complexity of providing accurate information. If a server does not list incompatible protocols, clients cannot gain authenticated information about their availability and so cannot detect downgrade attacks against those protocols.

During rollout of a new, incompatible protocol, until the deployment is stable and not at risk of being disabled, servers SHOULD NOT advertise the existence of the new protocol.

Protocol deployments that are in the process of being disabled first need to be removed from the incompatible_protocols extension. If a disabled protocol is advertised to clients, clients might regard this as a downgrade attack. Though the incompatible_protocols extension only applies at the time of the TLS handshake, clients might take some time to act on the information. If an incompatible protocol is removed from deployment between when the client completes a handshake and when it acts, this could be treated as an error by the client.

6. Security Considerations

This design depends on the integrity of the TLS handshake across all forms, including TLS [RFC8446], DTLS [DTLS], and QUIC [QUIC-TLS]. Similarly, integrity is necessary across all TLS versions that a client is willing to negotiate. An attacker that can modify a TLS handshake in any one of these protocols or versions can cause a client to believe that other options do not exist.

7. IANA Considerations

IANA is requested to assign a new value from the "TLS ExtensionType Values" registry:

Value: TBD

Extension Name: incompatible_protocols

TLS 1.3: CH, EE

DTLS-Only: N

Recommended: Y

Reference: this document, Section 4

8. References

8.1. Normative References

[ALPN] 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>.

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

[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>.

[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>.

8.2. Informative References

[BCP126] Abley, J. and K. Lindqvist, "Operation of Anycast Services", BCP 126, RFC 4786, December 2006. <https://www.rfc-editor.org/info/bcp126>

[DTLS] Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-dtls13-43, 30 April 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-tls-dtls13-43>.

[HTTP11] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-19, 12 September 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-messaging-19>.

[HTTP2] Thomson, M. and C. Benfield, "HTTP/2", Work in Progress, Internet-Draft, draft-ietf-httpbis-http2bis-07, 24 January 2022, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-http2bis-07>.

[HTTP3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-34, 2 February 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-http-34>.

[QUIC-TLS] Thomson, M. and S. Turner, "Using TLS to Secure QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-tls-34, 14 January 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-tls-34>.

[QUIC-VN] Schinazi, D. and E. Rescorla, "Compatible Version Negotiation for QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-version-negotiation-05, 25 October 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-version-negotiation-05>.

[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>.

[SVCB] Schwartz, B., Bishop, M., and E. Nygren, "Service binding and parameter specification via the DNS (DNS SVCB and HTTPSSVC)", Work in Progress, Internet-Draft, draft-ietf-dnsop-svcb-httpssvc-03, 11 June 2020, <https://datatracker.ietf.org/doc/html/draft-ietf-dnsop-svcb-httpssvc-03>.

[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/rfc/rfc8446>.

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

Appendix A. Acknowledgments

Benjamin Schwartz provided significant input into the design of the mechanism and helped simplify the overall design.

Appendix B. Defining Logical Servers

As incompatible protocols use different protocol stacks, they also use different endpoints. In other words, it is impossible for a single endpoint to support multiple incompatible protocols. Thus, it is necessary to understand the set of endpoints at a server that offer the incompatible protocols.

Thus, the definition of where incompatible protocols needs to encompass multiple endpoints somehow.

A number of choices are possible here:

* The set of endpoints that are authoritative for the same domain name.

* The set of endpoints that are authoritative for the same "authority" as defined in RFC 3986 [URI], which is in effect domain name plus port number.

* The set of endpoints that are referenced by the same SVCB ServiceMode record; see Section 2.4.3 of [SVCB].

* The set of endpoints that share an IP address.

* The set of endpoints that share an IP address and port number.

The challenge with options based on domain name is that it might prevent the use of multiple service providers. This is a common practice for HTTP, where the same domain name can be operated by multiple CDN operators.

Having multiple service operators also rules out using SVCB ServiceMode records also as different records might be used to identify different operators.

Hosts on the same IP address might work, but common deployment practices include use of different ports for entirely different services. These can have different operational constraints, such as deployment schedules. Including different ports in the same scope could force all services on the same host to support a consistent set of protocols.

This leaves IP and port. There is a risk that the same port number is used for completely different purposes depending on the choice of protocol. This practice is sufficiently rare that it is not anticipated to be a problem. Finally, a deployment with no ability to coordinate the deployment of protocols that share an IP and port can choose not to advertise the availability of incompatible protocols.

Appendix C. Incompatible Protocols and SVCB

The SVCB record [SVCB] allows a client to learn about services associated with a domain name. This includes how to locate a server, along with supplementary information about the server, including protocols that the server supports. This allows a client to start using a protocol of their choice without added latency, as a query for SVCB records can be performed at the same time as name resolution.

However, SVCB provides no protection against a downgrade attack between incompatible protocols. An attacker could remove DNS records for protocols that the client prefers, leaving the client to believe that only less-preferred options are available. If removed options are not compatible with the option that is chosen, the client will attempt those less-preferred options when attempting a TLS handshake.

Authenticating all of the information presented in SVCB records might provide clients with complete information about server support, but this is impractical for several reasons:

* it is not possible to ensure that all server instances in a deployment have the same protocol configuration, as deployments for a single name routinely include multiple providers that cannot coordinate closely;

* the ability to provide a subset of valid DNS records is integral to many strategies for managing servers; and

* ensuring that cached DNS records are synchronized with server state is challenging in a number of deployments.

Overall, an authenticated TLS handshake is a better source of authoritative information.

Author's Address

Martin Thomson
Mozilla

Email: mt@lowentropy.net