Network Working Group
Internet-Draft
Intended status: Informational
Expires: 7 January 2022

M. Thomson
Mozilla

6 July 2021

Secure Negotiation of Incompatible Protocols in TLS
draft-thomson-tls-snip-02

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.

Discussion Venues

This note is to be removed before publishing as an RFC.

Discussion of this document takes place on the TLS Working Group mailing list (tls@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/tls/.

Source for this draft and an issue tracker can be found at https://github.com/martinthomson/snip.

Status of This Memo

Table of Contents

1. Introduction 2
2. Terminology 3
3. Incompatible Protocols and SVCB 4
4. Authenticating Incompatible Protocols 4
5. Incompatible Protocol Selection 5
6. Logical Servers 6
6.1. Validation Process 7
6.2. QUIC Version Negotiation 7
6.3. Alternative Services 7
7. Operational Considerations 8
8. Security Considerations 8
9. IANA Considerations 9
10. References 9
10.1. Normative References 9
10.2. Informative References 9
Appendix A. Acknowledgments 10
Appendix B. Defining Logical Servers 11
Author's Address 11

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.

This downgrade protection 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.

This document defines an extension to TLS that allows clients to discover when servers support alternative protocols that are incompatible with the currently-selected TLS version. This might be used to avoid downgrade attack caused by interference in protocol discovery mechanisms.

This extension is motivated by the addition of new mechanisms, such as [SVCB]. SVCB enables the discovery of servers that support multiple different protocols, some of which are incompatible. The extension can also be used to authenticate protocol choices that are discovered by other means.

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 consider "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 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 the lookup can be performed concurrently with other name resolution. The added cost of the additional DNS queries is minimal.

However, SVCB provides no protection against a downgrade attack between incompatible protocols. An attacker could remove DNS records for client-preferred protocols, leaving the client to believe that only less-preferred options are available. If those options are not compatible with the client-preferred option, the client will not know to attempt these. The client then only offers options compatible with the less-preferred options when attempting a TLS handshake. Even if a client were to inform the server that it supports a more preferred protocol, the server would not be able to act upon it.

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

* it is difficult to ensure that cached DNS records are synchronized with server state.

Overall, an authenticated TLS handshake is a better source of authoritative information about the protocols that are supported by servers.

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 6 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 value server extension uses the "ProtocolName" type defined in [ALPN], which is repeated here. This syntax is shown in Figure 1.

```
opaque ProtocolName<1..2^8-1>;
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 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.

5. Incompatible Protocol Selection

This document expands the definition of protocol negotiation to include both compatible and incompatible protocols and provide protection against downgrade for both types of selection. ALPN [ALPN] only considers compatible protocols: the client presents a set of compatible options and the server chooses its most preferred.

With an selection of protocols that includes incompatible options, the client makes a selection between incompatible options before making a connection attempt. Therefore, this design does not enable negotiation, it instead provides the client with information about other incompatible protocols that the server might support.

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 about its options. What a client does with this information is left to client policy.

In brief:

* For compatible protocols, the client offers all acceptable options

and the server selects its most preferred

* For incompatible protocols, information the server offers is authenticated and the client is able to act on that

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 protocol, 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.

6. Logical Servers

The set of endpoints over which clients can assume availability of incompatible protocols is 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 servers need to be aware of all instances that might answer to the same IP address and port; see Section 7.

6.1. Validation Process

The type of protocol authentication scope describes how a client might learn of all of the service endpoints that a server offers in that scope. If a client has attempted to discover service endpoints using the methods defined by the protocol authentication scope, receiving an incompatible_protocols extension from a server is a strong indication of a potential downgrade attack.

A client considers that a downgrade attack might have occurred if a server advertises that there are endpoints that support a protocol that the client prefers over the protocol that is currently in use.

In response to detecting a potential downgrade attack, a client might abandon the current connection attempt and report an error. A client that supports discovery of incompatible protocols, but chooses not to make a discovery attempt under normal conditions might instead not fail, but it could use what it learns as cause to initiate discovery.

6.2. QUIC Version Negotiation

QUIC enables the definition of incompatible protocols that share a port. This mechanism can be used to authenticate the choice of application protocol in QUIC. QUIC version negotiation [QUIC-VN] is used to authenticate the choice of QUIC version.

As there are two potentially competing sets of preferences, clients need to set preferences for QUIC version and application protocol that do not result in inconsistent outcomes. 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 lead to a failure condition that cannot be reconciled.

6.3. Alternative Services

It is possible to negotiate protocols based on an established connection without exposure to downgrade. The Alternative Services [ALTSVC] bootstrapping in HTTP/3 [HTTP3] does just that. Assuming that HTTP/2 or HTTP/1.1 are not vulnerable to attacks that would compromise integrity, a server can advertise the presence of an endpoint that supports HTTP/3.

Under these assumptions Alternative Services is secure, but it has performance trade-offs. A client could attempt the protocol it prefers most, but that comes at a risk that this protocol is not supported by a server. A client could implement a fallback, which might even be performed concurrently (see [HAPPY-EYEBALLS]), but this

costs time and resources. A client avoids these costs by attempting

the protocol it believes to be most widely supported, though this comes with a performance penalty in cases where the most-preferred protocol is supported.

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

7. Operational Considerations

By listing incompatible protocols a server needs reliable knowledge of the existence of these alternatives. This depends on some coordination of deployments. In particular, coordination is important if a load balancer distributes load for a single IP address to multiple server instances. Ensuring consistent configuration of servers could present operational difficulties as it requires that incompatible protocols are only listed when those protocols are deployed across all server instances.

Server deployments can choose not to provide information about incompatible protocols, which denies clients information about downgrade attacks but might avoid the operational complexity of providing accurate information.

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 disabled, first need to be removed from the incompatible_protocols extension or there could be some loss of service. 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.

If a server does not list incompatible protocols, clients cannot learn about other services and so cannot detect downgrade attacks against those protocols.

8. Security Considerations

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

9. IANA Considerations

TODO: register the extension

10. References

10.1. 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>.

[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>.

10.2. Informative References

[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>.

[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>.

[HAPPY-EYEBALLS] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, DOI 10.17487/RFC6555, April 2012, <https://www.rfc-editor.org/rfc/rfc6555>.

[HTTP11] Fielding, R. T., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-16, 27 May 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-httpbis-messaging-16>.

[HTTP2] 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/rfc/rfc7540>.

[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-04, 26 May 2021, <https://datatracker.ietf.org/doc/html/draft-ietf-quic-version-negotiation-04>.

[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 in many cases impossible for the exactly same endpoint to support multiple incompatible protocols. Thus, it is necessary to understand the set of endpoints at a server that offer the incompatible protocols.

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.

* 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, which 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 always a risk that the same port number is used for completely different purposes depending on the choice of protocol, but this practice is sufficiently rare that it is not anticipated to be a problem.

Author's Address

Martin Thomson
Mozilla

Email: mt@lowentropy.net