Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 28 December 2020

R. Barnes
Cisco
S. Iyengar
Facebook
N. Sullivan
Cloudflare
E. Rescorla
Mozilla

26 June 2020

Delegated Credentials for TLS
draft-ietf-tls-subcerts-09

Abstract

The organizational separation between the operator of a TLS endpoint and the certification authority can create limitations. For example, the lifetime of certificates, how they may be used, and the algorithms they support are ultimately determined by the certification authority. This document describes a mechanism by which operators may delegate their own credentials for use in TLS, without breaking compatibility with peers that do not support this specification.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/tlswg/tls-subcerts (https://github.com/tlswg/tls-subcerts).

Status of This Memo

This Internet-Draft will expire on 28 December 2020.

Copyright Notice

Table of Contents

1. Introduction

Typically, a TLS server uses a certificate provided by some entity other than the operator of the server (a "Certification Authority" or CA) [RFC8446] [RFC5280]. This organizational separation makes the TLS server operator dependent on the CA for some aspects of its operations, for example:

* Whenever the server operator wants to deploy a new certificate, it has to interact with the CA.

* The server operator can only use TLS signature schemes for which the CA will issue credentials.

These dependencies cause problems in practice. Server operators often deploy TLS termination services in locations such as remote data centers or Content Delivery Networks (CDNs) where it may be difficult to detect key compromises. Short-lived certificates may be used to limit the exposure of keys in these cases.

However, short-lived certificates need to be renewed more frequently than long-lived certificates. If an external CA is unable to issue a certificate in time to replace a deployed certificate, the server would no longer be able to present a valid certificate to clients. With short-lived certificates, there is a smaller window of time to renew a certificates and therefore a higher risk that an outage at a CA will negatively affect the uptime of the service.

To reduce the dependency on external CAs, this document proposes a limited delegation mechanism that allows a TLS peer to issue its own credentials within the scope of a certificate issued by an external CA. These credentials only enable the recipient of the delegation to speak for names that the CA has authorized. For clarity, we will refer to the certificate issued by the CA as a "certificate", or "delegation certificate", and the one issued by the operator as a "delegated credential" or "DC".

2. Conventions and 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.

2.1. Change Log

(*) indicates changes to the wire protocol.

draft-09

* Address case nits

* Fix section bullets in 4.1.3.

* Add operational considerations section for clock skew

* Add text around using an oracle to forge DCs in the future and past

* Add text about certificate extension vs EKU

draft-08

* Include details about the impact of signature forgery attacks

* Copy edits

* Fix section about DC reuse

* Incorporate feedback from Jonathan Hammell and Kevin Jacobs on the list

draft-07

* Minor text improvements

draft-06

* Modified IANA section, fixed nits

draft-05

* Removed support for PKCS 1.5 RSA signature algorithms.

* Additional security considerations.

draft-04

* Add support for client certificates.

draft-03

- Remove protocol version from the Credential structure. (*)

draft-02

- Change public key type. (*)

- Change DelegationUsage extension to be NULL and define its object identifier.

- Drop support for TLS 1.2.

- Add the protocol version and credential signature algorithm to the Credential structure. (*)

- Specify undefined behavior in a few cases: when the client receives a DC without indicated support; when the client indicates the extension in an invalid protocol version; and when DCs are sent as extensions to certificates other than the end-entity certificate.

3. Solution Overview

A delegated credential is a digitally signed data structure with two semantic fields: a validity interval and a public key (along with its associated signature algorithm). The signature on the credential indicates a delegation from the certificate that is issued to the peer. The private key used to sign a credential corresponds to the public key of the peer's X.509 end-entity certificate [RFC5280].

A TLS handshake that uses delegated credentials differs from a standard handshake in a few important ways:

- The initiating peer provides an extension in its ClientHello or CertificateRequest that indicates support for this mechanism.

- The peer sending the Certificate message provides both the certificate chain terminating in its certificate as well as the delegated credential.

- The authenticating initiator uses information from the peer's certificate to verify the delegated credential and that the peer is asserting an expected identity.

- Peers accepting the delegated credential use it as the certificate key for the TLS handshake

As detailed in Section 4, the delegated credential is cryptographically bound to the end-entity certificate with which the credential may be used. This document specifies the use of delegated credentials in TLS 1.3 or later; their use in prior versions of the protocol is not allowed.

Delegated credentials allow a peer to terminate TLS connections on behalf of the certificate owner. If a credential is stolen, there is no mechanism for revoking it without revoking the certificate itself. To limit exposure in case of delegated credential private key compromise, delegated credentials have a maximum validity period. In the absence of an application profile standard specifying otherwise, the maximum validity period is set to 7 days. Peers MUST NOT issue credentials with a validity period longer than the maximum validity period. This mechanism is described in detail in Section 4.1.

It was noted in [XPROT] that certificates in use by servers that support outdated protocols such as SSLv2 can be used to forge signatures for certificates that contain the keyEncipherment KeyUsage ([RFC5280] section 4.2.1.3). In order to prevent this type of cross-protocol attack, we define a new DelegationUsage extension to X.509 that permits use of delegated credentials. (See Section 4.2.)

3.1. Rationale

Delegated credentials present a better alternative than other delegation mechanisms like proxy certificates [RFC3820] for several reasons:

* There is no change needed to certificate validation at the PKI layer.

* X.509 semantics are very rich. This can cause unintended consequences if a service owner creates a proxy certificate where the properties differ from the leaf certificate. For this reason, delegated credentials have very restricted semantics that should not conflict with X.509 semantics.

* Proxy certificates rely on the certificate path building process to establish a binding between the proxy certificate and the server certificate. Since the certificate path building process is not cryptographically protected, it is possible that a proxy certificate could be bound to another certificate with the same public key, with different X.509 parameters. Delegated credentials, which rely on a cryptographic binding between the entire certificate and the delegated credential, cannot.

* Each delegated credential is bound to a specific signature algorithm that may be used to sign the TLS handshake ([RFC8446]

section 4.2.3). This prevents them from being used with other, perhaps unintended signature algorithms.

3.2. Related Work

Many of the use cases for delegated credentials can also be addressed using purely server-side mechanisms that do not require changes to client behavior (e.g., a PKCS#11 interface or a remote signing mechanism [KEYLESS]). These mechanisms, however, incur per-transaction latency, since the front-end server has to interact with a back-end server that holds a private key. The mechanism proposed in this document allows the delegation to be done off-line, with no per-transaction latency. The figure below compares the message flows for these two mechanisms with TLS 1.3 [RFC8446].

Remote key signing:

```
   Client            Front-End            Back-End
     |----ClientHello--->|                    |
     |<---ServerHello----|                    |
     |<---Certificate----|                    |
     |                   |<---remote sign---->|
     |<---CertVerify-----|                    |
     |        ...        |                    |
```

Delegated credentials:

```
   Client            Front-End            Back-End
     |                   |<--DC distribution->|
     |----ClientHello--->|                    |
     |<---ServerHello----|                    |
     |<---Certificate----|                    |
     |<---CertVerify-----|                    |
     |        ...        |                    |
```

These two mechanisms can be complementary. A server could use credentials for clients that support them, while using [KEYLESS] to support legacy clients. The private key for a delegated credential can be used in place of a certificate private key, so it is important that the Front-End and Back-End are parties that have a trusted relationship.

Use of short-lived certificates with automated certificate issuance, e.g., with Automated Certificate Managment Environment (ACME) [RFC8555], reduces the risk of key compromise, but has several limitations. Specifically, it introduces an operationally-critical dependency on an external party. It also limits the types of algorithms supported for TLS authentication to those the CA is willing to issue a certificate for. Nonetheless, existing automated issuance APIs like ACME may be useful for provisioning delegated credentials.

4. Delegated Credentials

While X.509 forbids end-entity certificates from being used as issuers for other certificates, it is valid to use them to issue other signed objects as long as the certificate contains the digitalSignature KeyUsage ([RFC5280] section 4.2.1.3). We define a new signed object format that would encode only the semantics that are needed for this application. The credential has the following structure:

```
   struct {
     uint32 valid_time;
     SignatureScheme expected_cert_verify_algorithm;
     opaque ASN1_subjectPublicKeyInfo<1..2^24-1>;
   } Credential;
```

valid_time: Time in seconds relative to the beginning of the delegation certificate's notBefore value after which the delegated credential is no longer valid. This MUST NOT exceed 7 days.

expected_cert_verify_algorithm: The signature algorithm of the credential key pair, where the type SignatureScheme is as defined in [RFC8446]. This is expected to be the same as CertificateVerify.algorithm sent by the server. Only signature algorithms allowed for use in CertificateVerify messages are allowed. When using RSA, the public key MUST NOT use the rsaEncryption OID, as a result, the following algorithms are not allowed for use with delegated credentials: rsa_pss_rsae_sha256, rsa_pss_rsae_sha384, rsa_pss_rsae_sha512.

ASN1_subjectPublicKeyInfo: The credential's public key, a DER-encoded [X.690] SubjectPublicKeyInfo as defined in [RFC5280].

The delegated credential has the following structure:

```
   struct {
     Credential cred;
     SignatureScheme algorithm;
     opaque signature<0..2^16-1>;
   } DelegatedCredential;
```

algorithm: The signature algorithm used to verify DelegatedCredential.signature.

signature: The delegation, a signature that binds the credential to the end-entity certificate’s public key as specified below. The signature scheme is specified by DelegatedCredential.algorithm.

The signature of the DelegatedCredential is computed over the concatenation of:

1. A string that consists of octet 32 (0x20) repeated 64 times.

2. The context string "TLS, server delegated credentials" for servers and "TLS, client delegated credentials" for clients.

3. A single 0 byte, which serves as the separator.

4. The DER-encoded X.509 end-entity certificate used to sign the DelegatedCredential.

5. DelegatedCredential.cred.

6. DelegatedCredential.algorithm.

The signature effectively binds the credential to the parameters of the handshake in which it is used. In particular, it ensures that credentials are only used with the certificate and signature algorithm chosen by the delegator.

The code changes required in order to create and verify delegated credentials, and the implementation complexity this entails, are localized to the TLS stack. This has the advantage of avoiding changes to security-critical and often delicate PKI code.

4.1. Client and Server Behavior

This document defines the following TLS extension code point.

```
enum {
  ...
  delegated_credential(34),
  (65535)
} ExtensionType;
```

4.1.1. Server Authentication

A client which supports this specification SHALL send a "delegated_credential" extension in its ClientHello. The body of the extension consists of a SignatureSchemeList:

```
      struct {
        SignatureScheme supported_signature_algorithm<2..2^16-2>;
      } SignatureSchemeList;
```

If the client receives a delegated credential without indicating support, then the client MUST abort with an "unexpected_message" alert.

If the extension is present, the server MAY send a delegated credential; if the extension is not present, the server MUST NOT send a delegated credential. The server MUST ignore the extension unless TLS 1.3 or a later version is negotiated.

The server MUST send the delegated credential as an extension in the CertificateEntry of its end-entity certificate; the client SHOULD ignore delegated credentials sent as extensions to any other certificate.

The expected_cert_verify_algorithm field MUST be of a type advertised by the client in the SignatureSchemeList and is considered invalid otherwise. Clients that receive invalid delegated credentials MUST terminate the connection with an "illegal_parameter" alert.

4.1.2. Client Authentication

A server that supports this specification SHALL send a "delegated_credential" extension in the CertificateRequest message when requesting client authentication. The body of the extension consists of a SignatureSchemeList. If the server receives a delegated credential without indicating support in its CertificateRequest, then the server MUST abort with an "unexpected_message" alert.

If the extension is present, the client MAY send a delegated credential; if the extension is not present, the client MUST NOT send a delegated credential. The client MUST ignore the extension unless TLS 1.3 or a later version is negotiated.

The client MUST send the delegated credential as an extension in the CertificateEntry of its end-entity certificate; the server SHOULD ignore delegated credentials sent as extensions to any other certificate.

The algorithm field MUST be of a type advertised by the server in the "signature_algorithms" extension of the CertificateRequest message and the expected_cert_verify_algorithm field MUST be of a type advertised by the server in the SignatureSchemeList and considered invalid otherwise. Servers that receive invalid delegated credentials MUST terminate the connection with an "illegal_parameter" alert.

4.1.3. Validating a Delegated Credential

On receiving a delegated credential and a certificate chain, the peer validates the certificate chain and matches the end-entity certificate to the peer's expected identity. It also takes the following steps:

1. Verify that the current time is within the validity interval of the credential. This is done by asserting that the current time is no more than the delegation certificate's notBefore value plus DelegatedCredential.cred.valid_time.

2. Verify that the credential's remaining validity time is no more than the maximum validity period. This is done by asserting that the current time is no more than the delegation certificate's notBefore value plus DelegatedCredential.cred.valid_time plus the maximum validity period.

3. Verify that expected_cert_verify_algorithm matches the scheme indicated in the peer's CertificateVerify message and that the algorithm is allowed for use with delegated credentials.

4. Verify that the end-entity certificate satisfies the conditions in Section 4.2.

5. Use the public key in the peer's end-entity certificate to verify the signature of the credential using the algorithm indicated by DelegatedCredential.algorithm.

If one or more of these checks fail, then the delegated credential is deemed invalid. Clients and servers that receive invalid delegated credentials MUST terminate the connection with an "illegal_parameter" alert. If successful, the participant receiving the Certificate message uses the public key in the credential to verify the signature in the peer's CertificateVerify message.

4.2. Certificate Requirements

We define a new X.509 extension, DelegationUsage, to be used in the certificate when the certificate permits the usage of delegated credentials. What follows is the ASN.1 [X.680] for the DelegationUsage certificate extension.

```
    ext-delegationUsage EXTENSION  ::= {
        SYNTAX DelegationUsage IDENTIFIED BY id-ce-delegationUsage
    }

    DelegationUsage ::= NULL

    id-ce-delegationUsage OBJECT IDENTIFIER ::=
        { iso(1) identified-organization(3) dod(6) internet(1)
          private(4) enterprise(1) id-cloudflare(44363) 44 }
```

The extension MUST be marked non-critical. (See Section 4.2 of [RFC5280].) The client MUST NOT accept a delegated credential unless the server's end-entity certificate satisfies the following criteria:

* It has the DelegationUsage extension.

* It has the digitalSignature KeyUsage (see the KeyUsage extension defined in [RFC5280]).

A new extension was chosen instead of adding a new Extended Key Usage (EKU) to be compatible with deployed TLS and PKI software stacks without requiring CAs to issue new intermediate certificates.

5. Operational Considerations

5.1. Client Clock Skew

One of the risks of deploying a short-lived credential system based on absolute time is client clock skew. If a client's clock is sufficiently ahead or behind of the server's clock, then clients will reject credentials that are valid from the server's perspective. Clock skew also affects the validity of the original certificates. The lifetime of the delegated credential should be set taking clock skew into account. Clock skew may affect a delegated credential at the beginning and end of its validity periods, which should also be taken into account.

6. IANA Considerations

This document registers the "delegated_credentials" extension in the "TLS ExtensionType Values" registry. The "delegated_credentials" extension has been assigned a code point of 34. The IANA registry lists this extension as "Recommended" (i.e., "Y") and indicates that it may appear in the ClientHello (CH), CertificateRequest (CR), or Certificate (CT) messages in TLS 1.3 [RFC8446].

This document also defines an ASN.1 module for the DelegationUsage certificate extension in Appendix A. IANA is requested to register an Object Identfier (OID) for the ASN.1 in "SMI Security for PKIX Module Identifier" arc. An OID for the DelegationUsage certificate extension is not needed as it is already assigned to the extension from Cloudflare's IANA Private Enterprise Number (PEN) arc.

7. Security Considerations

7.1. Security of Delegated Credential's Private Key

Delegated credentials limit the exposure of the private key used in a TLS connection by limiting its validity period. An attacker who compromises the private key of a delegated credential can act as a man-in-the-middle until the delegated credential expires. However, they cannot create new delegated credentials. Thus, delegated credentials should not be used to send a delegation to an untrusted party, but is meant to be used between parties that have some trust relationship with each other. The secrecy of the delegated credential's private key is thus important and access control mechanisms SHOULD be used to protect it, including file system controls, physical security, or hardware security modules.

7.2. Re-use of Delegated Credentials in Multiple Contexts

It is not possible to use the same delegated credential for both client and server authentication because issuing parties compute the corresponding signature using a context string unique to the intended role (client or server).

7.3. Revocation of Delegated Credentials

Delegated credentials do not provide any additional form of early revocation. Since it is short lived, the expiry of the delegated credential would revoke the credential. Revocation of the long term private key that signs the delegated credential also implicitly revokes the delegated credential.

7.4. Interactions with Session Resumption

If a client decides to cache the certificate chain and re-validate it when resuming a connection, the client SHOULD also cache the associated delegated credential and re-validate it.

7.5. Privacy Considerations

Delegated credentials can be valid for 7 days and it is much easier for a service to create delegated credential than a certificate signed by a CA. A service could determine the client time and clock skew by creating several delegated credentials with different expiry timestamps and observing whether the client would accept it. Client time could be unique and thus privacy sensitive clients, such as browsers in incognito mode, who do not trust the service might not want to advertise support for delegated credentials or limit the number of probes that a server can perform.

7.6. The Impact of Signature Forgery Attacks

When TLS 1.2 servers support RSA key exchange, they may be vulnerable to attacks that allow forging an RSA signature over an arbitrary message [BLEI]. TLS 1.2 [RFC5246] (Section 7.4.7.1.) describes a mitigation strategy requiring careful implementation of timing resistant countermeasures for preventing these attacks. Experience shows that in practice, server implementations may fail to fully stop these attacks due to the complexity of this mitigation [ROBOT]. For TLS 1.2 servers that support RSA key exchange using a DC-enabled end-entity certificate, a hypothetical signature forgery attack would allow forging a signature over a delegated credential. The forged credential could then be used by the attacker as the equivalent of a man-in-the-middle certificate, valid for 7 days.

Server operators should therefore minimize the risk of using DC-enabled end-entity certificates where a signature forgery oracle may be present. If possible, server operators may choose to use DC-enabled certificates only for signing credentials, and not for serving non-DC TLS traffic. Furthermore, server operators may use elliptic curve certificates for DC-enabled traffic, while using RSA certificates without the DelegationUsage certificate extension for non-DC traffic; this completely prevents such attacks.

Note that if a signature can be forged over an arbitrary credential, the attacker can choose any value for the valid_time field. Repeated signature forgeries therefore allow the attacker to create multiple delegated credentials that can cover the entire validity period of the certificate. Temporary exposure of the key or a signing oracle may allow the attacker to impersonate a server for the lifetime of the certificate.

8. Acknowledgements

Thanks to David Benjamin, Christopher Patton, Kyle Nekritz, Anirudh Ramachandran, Benjamin Kaduk, Kazuho Oku, Daniel Kahn Gillmor, Watson Ladd, Robert Merget, Juraj Somorovsky, Nimrod Aviram for their discussions, ideas, and bugs they have found.

9. References

9.1. Normative References

[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>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[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>.

[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/info/rfc8446>.

[X.680] ITU-T, "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ISO/IEC 8824-1:2015, November 2015.

[X.690] ITU-T, "Information technology - ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ISO/IEC 8825-1:2015, November 2015.

9.2. Informative References

[BLEI] Bleichenbacher, D., "Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS #1", Advances in Cryptology -- CRYPTO'98, LNCS vol. 1462, pages: 1-12 , 1998.

[KEYLESS] Sullivan, N. and D. Stebila, "An Analysis of TLS Handshake Proxying", IEEE Trustcom/BigDataSE/ISPA 2015 , 2015.

[RFC3820] Tuecke, S., Welch, V., Engert, D., Pearlman, L., and M. Thompson, "Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile", RFC 3820, DOI 10.17487/RFC3820, June 2004, <https://www.rfc-editor.org/info/rfc3820>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <https://www.rfc-editor.org/info/rfc5912>.

[RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <https://www.rfc-editor.org/info/rfc8555>.

[ROBOT] Boeck, H., Somorovsky, J., and C. Young, "Return Of Bleichenbacher's Oracle Threat (ROBOT)", 27th USENIX Security Symposium , 2018.

[XPROT] Jager, T., Schwenk, J., and J. Somorovsky, "On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption", Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security , 2015.

Appendix A. ASN.1 Module

The following ASN.1 module provides the complete definition of the DelegationUsage certificate extension. The ASN.1 module makes imports from [RFC5912].

```
   DelegatedCredentialExtn
     { iso(1) identified-organization(3) dod(6) internet(1)
       security(5) mechanisms(5) pkix(7) id-mod(0)
       id-mod-delegated-credential-extn(TBD) }

   DEFINITIONS IMPLICIT TAGS ::=
   BEGIN

   -- EXPORT ALL

   IMPORTS

   EXTENSION
     FROM PKIX-CommonTypes-2009 -- From RFC 5912
     { iso(1) identified-organization(3) dod(6) internet(1)
       security(5) mechanisms(5) pkix(7) id-mod(0)
       id-mod-pkixCommon-02(57) } ;

   -- OID

   id-cloudflare OBJECT IDENTIFIER ::=
     { iso(1) identified-organization(3) dod(6) internet(1) private(4)
       enterprise(1) 44363 }

   -- EXTENSION

   ext-delegationUsage EXTENSION ::=
     { SYNTAX DelegationUsage
       IDENTIFIED BY id-ce-delegationUsage }

   id-ce-delegationUsage OBJECT IDENTIFIER ::= { id-cloudflare 44 }

   DelegationUsage ::= NULL

   END
```

Authors’ Addresses

Richard Barnes
Cisco

Email: rlb@ipv.sx

Subodh Iyengar
Facebook

Email: subodh@fb.com

Nick Sullivan
Cloudflare

Email: nick@cloudflare.com

Eric Rescorla
Mozilla

Email: ekr@rtfm.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 28 July 2021

R. Barnes
Cisco
S. Iyengar
Facebook
N. Sullivan
Cloudflare
E. Rescorla
Mozilla

24 January 2021

Delegated Credentials for TLS
draft-ietf-tls-subcerts-10

Abstract

The organizational separation between the operator of a TLS endpoint and the certification authority can create limitations. For example, the lifetime of certificates, how they may be used, and the algorithms they support are ultimately determined by the certification authority. This document describes a mechanism by which operators may delegate their own credentials for use in TLS, without breaking compatibility with peers that do not support this specification.

Discussion Venues

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

Source for this draft and an issue tracker can be found at https://github.com/tlswg/tls-subcerts (https://github.com/tlswg/tls-subcerts).

Status of This Memo

This Internet-Draft will expire on 28 July 2021.

Copyright Notice

Table of Contents

1. Introduction

Typically, a TLS server uses a certificate provided by some entity other than the operator of the server (a "Certification Authority" or CA) [RFC8446] [RFC5280]. This organizational separation makes the TLS server operator dependent on the CA for some aspects of its operations, for example:

* Whenever the server operator wants to deploy a new certificate, it has to interact with the CA.

* The server operator can only use TLS signature schemes for which the CA will issue credentials.

These dependencies cause problems in practice. Server operators often deploy TLS termination services in locations such as remote data centers or Content Delivery Networks (CDNs) where it may be difficult to detect key compromises. Short-lived certificates may be used to limit the exposure of keys in these cases.

However, short-lived certificates need to be renewed more frequently than long-lived certificates. If an external CA is unable to issue a certificate in time to replace a deployed certificate, the server would no longer be able to present a valid certificate to clients. With short-lived certificates, there is a smaller window of time to renew a certificates and therefore a higher risk that an outage at a CA will negatively affect the uptime of the service.

To reduce the dependency on external CAs, this document proposes a limited delegation mechanism that allows a TLS peer to issue its own credentials within the scope of a certificate issued by an external CA. These credentials only enable the recipient of the delegation to speak for names that the CA has authorized. Furthermore, this mechanism allows the server to use modern signature algorithms such as Ed25519 [RFC8032] even if their CA does not support them.

We will refer to the certificate issued by the CA as a "certificate", or "delegation certificate", and the one issued by the operator as a "delegated credential" or "DC".

2. Conventions and 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.

2.1. Change Log

RFC EDITOR PLEASE DELETE THIS SECTION.

(*) indicates changes to the wire protocol.

draft-10 * Address superficial comments * Add example certificate

draft-09

- Address case nits
- Fix section bullets in 4.1.3.
- Add operational considerations section for clock skew
- Add text around using an oracle to forge DCs in the future and past
- Add text about certificate extension vs EKU

draft-08

- Include details about the impact of signature forgery attacks
- Copy edits
- Fix section about DC reuse
- Incorporate feedback from Jonathan Hammell and Kevin Jacobs on the list

draft-07

- Minor text improvements

draft-06

- Modified IANA section, fixed nits

draft-05

- Removed support for PKCS 1.5 RSA signature algorithms.
- Additional security considerations.

draft-04

- Add support for client certificates.

draft-03

- Remove protocol version from the Credential structure. (*)

draft-02

- Change public key type. (*)

- Change DelegationUsage extension to be NULL and define its object identifier.

- Drop support for TLS 1.2.

- Add the protocol version and credential signature algorithm to the Credential structure. (*)

- Specify undefined behavior in a few cases: when the client receives a DC without indicated support; when the client indicates the extension in an invalid protocol version; and when DCs are sent as extensions to certificates other than the end-entity certificate.

3. Solution Overview

A delegated credential (DC) is a digitally signed data structure with two semantic fields: a validity interval and a public key (along with its associated signature algorithm). The signature on the delegated credential indicates a delegation from the certificate that is issued to the peer. The private key used to sign a credential corresponds to the public key of the peer's X.509 end-entity certificate [RFC5280].

A TLS handshake that uses delegated credentials differs from a standard handshake in a few important ways:

- The initiating peer provides an extension in its ClientHello or CertificateRequest that indicates support for this mechanism.

- The peer sending the Certificate message provides both the certificate chain terminating in its certificate as well as the delegated credential.

- The authenticating initiator uses information from the peer's certificate to verify the delegated credential and that the peer is asserting an expected identity.

- Peers accepting the delegated credential use it as the certificate key for the TLS handshake

As detailed in Section 4, the delegated credential is cryptographically bound to the end-entity certificate with which the credential may be used. This document specifies the use of delegated credentials in TLS 1.3 or later; their use in prior versions of the protocol is not allowed.

Delegated credentials allow a peer to terminate TLS connections on behalf of the certificate owner. If a credential is stolen, there is no mechanism for revoking it without revoking the certificate itself. To limit exposure in case of the compromise of a delegated credential's private key, delegated credentials have a maximum validity period. In the absence of an application profile standard specifying otherwise, the maximum validity period is set to 7 days. Peers MUST NOT issue credentials with a validity period longer than the maximum validity period. This mechanism is described in detail in Section 4.1.

It was noted in [XPROT] that certificates in use by servers that support outdated protocols such as SSLv2 can be used to forge signatures for certificates that contain the keyEncipherment KeyUsage ([RFC5280] section 4.2.1.3). In order to prevent this type of cross-protocol attack, we define a new DelegationUsage extension to X.509 that permits use of delegated credentials. (See Section 4.2.)

3.1. Rationale

Delegated credentials present a better alternative than other delegation mechanisms like proxy certificates [RFC3820] for several reasons:

- There is no change needed to certificate validation at the PKI layer.

- X.509 semantics are very rich. This can cause unintended consequences if a service owner creates a proxy certificate where the properties differ from the leaf certificate. For this reason, delegated credentials have very restricted semantics that should not conflict with X.509 semantics.

- Proxy certificates rely on the certificate path building process to establish a binding between the proxy certificate and the server certificate. Since the certificate path building process is not cryptographically protected, it is possible that a proxy certificate could be bound to another certificate with the same public key, with different X.509 parameters. Delegated credentials, which rely on a cryptographic binding between the entire certificate and the delegated credential, cannot.

* Each delegated credential is bound to a specific signature algorithm for use use in the TLS handshake ([RFC8446] section 4.2.3). This prevents them from being used with other, perhaps unintended signature algorithms.

3.2. Related Work

Many of the use cases for delegated credentials can also be addressed using purely server-side mechanisms that do not require changes to client behavior (e.g., a PKCS#11 interface or a remote signing mechanism [KEYLESS]). These mechanisms, however, incur per-transaction latency, since the front-end server has to interact with a back-end server that holds a private key. The mechanism proposed in this document allows the delegation to be done off-line, with no per-transaction latency. The figure below compares the message flows for these two mechanisms with TLS 1.3 [RFC8446].

Remote key signing:

```
Client            Front-End            Back-End
  |----ClientHello--->|                    |
  |<---ServerHello----|                    |
  |<---Certificate----|                    |
  |                   |<---remote sign---->|
  |<---CertVerify-----|                    |
  |        ...        |                    |
```

Delegated Credential:

```
Client            Front-End            Back-End
  |                   |<--DC distribution->|
  |----ClientHello--->|                    |
  |<---ServerHello----|                    |
  |<---Certificate----|                    |
  |<---CertVerify-----|                    |
  |        ...        |                    |
```

These two mechanisms can be complementary. A server could use delegated credentials for clients that support them, while using [KEYLESS] to support legacy clients. The private key for a delegated credential can be used in place of a certificate private key, so it is important that the Front-End and Back-End are parties with a trusted relationship.

Use of short-lived certificates with automated certificate issuance, e.g., with Automated Certificate Management Environment (ACME) [RFC8555], reduces the risk of key compromise, but has several limitations. Specifically, it introduces an operationally-critical dependency on an external party (the CA). It also limits the types of algorithms supported for TLS authentication to those the CA is willing to issue a certificate for. Nonetheless, existing automated issuance APIs like ACME may be useful for provisioning delegated credentials.

4. Delegated Credentials

While X.509 forbids end-entity certificates from being used as issuers for other certificates, it is valid to use them to issue other signed objects as long as the certificate contains the digitalSignature KeyUsage ([RFC5280] section 4.2.1.3). We define a new signed object format that would encode only the semantics that are needed for this application. The Credential has the following structure:

```
struct {
  uint32 valid_time;
  SignatureScheme expected_cert_verify_algorithm;
  opaque ASN1_subjectPublicKeyInfo<1..2^24-1>;
} Credential;
```

valid_time: Time in seconds relative to the beginning of the delegation certificate's notBefore value after which the delegated credential is no longer valid. Endpoints will reject delegate credentials with valid_times exceeding 7 days (as described in Section 4.1).

expected_cert_verify_algorithm: The signature algorithm of the Credential key pair, where the type SignatureScheme is as defined in [RFC8446]. This is expected to be the same as the sender's CertificateVerify.algorithm. Only signature algorithms allowed for use in CertificateVerify messages are allowed. When using RSA, the public key MUST NOT use the rsaEncryption OID. As a result, the following algorithms are not allowed for use with delegated credentials: rsa_pss_rsae_sha256, rsa_pss_rsae_sha384, rsa_pss_rsae_sha512.

ASN1_subjectPublicKeyInfo: The Credential's public key, a DER-encoded [X.690] SubjectPublicKeyInfo as defined in [RFC5280].

The DelegatedCredential has the following structure:

```
      struct {
        Credential cred;
        SignatureScheme algorithm;
        opaque signature<0..2^16-1>;
      } DelegatedCredential;
```

cred: The Credential structure as previously defined.

algorithm: The signature algorithm used to verify DelegatedCredential.signature.

signature: The delegation, a signature that binds the credential to the end-entity certificate's public key as specified below. The signature scheme is specified by DelegatedCredential.algorithm.

The signature of the DelegatedCredential is computed over the concatenation of:

1. A string that consists of octet 32 (0x20) repeated 64 times.

2. The context string "TLS, server delegated credentials" for server authentication and "TLS, client delegated credentials" for client authentication.

3. A single 0 byte, which serves as the separator.

4. The DER-encoded X.509 end-entity certificate used to sign the DelegatedCredential.

5. DelegatedCredential.cred.

6. DelegatedCredential.algorithm.

The signature is computed by using the private key of the peer's end-entity certificate, with the algorithm indicated by DelegatedCredential.algorithm.

The signature effectively binds the credential to the parameters of the handshake in which it is used. In particular, it ensures that credentials are only used with the certificate and signature algorithm chosen by the delegator.

The code changes required in order to create and verify delegated credentials, and the implementation complexity this entails, are localized to the TLS stack. This has the advantage of avoiding changes to security-critical and often delicate PKI code.

4.1. Client and Server Behavior

This document defines the following TLS extension code point.

```
   enum {
     ...
     delegated_credential(34),
     (65535)
   } ExtensionType;
```

4.1.1. Server Authentication

A client which supports this specification SHALL send a "delegated_credential" extension in its ClientHello. The body of the extension consists of a SignatureSchemeList (defined in [RFC8446]):

```
   struct {
     SignatureScheme supported_signature_algorithm<2..2^16-2>;
   } SignatureSchemeList;
```

If the client receives a delegated credential without indicating support, then the client MUST abort with an "unexpected_message" alert.

If the extension is present, the server MAY send a delegated credential; if the extension is not present, the server MUST NOT send a delegated credential. The server MUST ignore the extension unless TLS 1.3 or a later version is negotiated.

The server MUST send the delegated credential as an extension in the CertificateEntry of its end-entity certificate; the client SHOULD ignore delegated credentials sent as extensions to any other certificate.

The expected_cert_verify_algorithm field MUST be of a type advertised by the client in the SignatureSchemeList and is considered invalid otherwise. Clients that receive invalid delegated credentials MUST terminate the connection with an "illegal_parameter" alert.

4.1.2. Client Authentication

A server that supports this specification SHALL send a "delegated_credential" extension in the CertificateRequest message when requesting client authentication. The body of the extension consists of a SignatureSchemeList. If the server receives a delegated credential without indicating support in its CertificateRequest, then the server MUST abort with an "unexpected_message" alert.

If the extension is present, the client MAY send a delegated credential; if the extension is not present, the client MUST NOT send a delegated credential. The client MUST ignore the extension unless TLS 1.3 or a later version is negotiated.

The client MUST send the delegated credential as an extension in the CertificateEntry of its end-entity certificate; the server SHOULD ignore delegated credentials sent as extensions to any other certificate.

The algorithm field MUST be of a type advertised by the server in the "signature_algorithms" extension of the CertificateRequest message and the expected_cert_verify_algorithm field MUST be of a type advertised by the server in the SignatureSchemeList and considered invalid otherwise. Servers that receive invalid delegated credentials MUST terminate the connection with an "illegal_parameter" alert.

4.1.3. Validating a Delegated Credential

On receiving a delegated credential and a certificate chain, the peer validates the certificate chain and matches the end-entity certificate to the peer's expected identity. It also takes the following steps:

1. Validate that DelegatedCredential.cred.valid_time is no more than 7 days.

2. Verify that the current time is within the validity interval of the credential. This is done by asserting that the current time is no more than the delegation certificate's notBefore value plus DelegatedCredential.cred.valid_time.

3. Verify that the delegated credential's remaining validity time is no more than the maximum validity period. This is done by asserting that the current time is no more than the delegation certificate's notBefore value plus DelegatedCredential.cred.valid_time plus the maximum validity period.

4. Verify that expected_cert_verify_algorithm matches the scheme indicated in the peer's CertificateVerify message and that the algorithm is allowed for use with delegated credentials.

5. Verify that the end-entity certificate satisfies the conditions in Section 4.2.

6. Use the public key in the peer’s end-entity certificate to verify the signature of the credential using the algorithm indicated by DelegatedCredential.algorithm.

If one or more of these checks fail, then the delegated credential is deemed invalid. Clients and servers that receive invalid delegated credentials MUST terminate the connection with an "illegal_parameter" alert.

If successful, the participant receiving the Certificate message uses the public key in DelegatedCredential.cred to verify the signature in the peer’s CertificateVerify message.

4.2. Certificate Requirements

We define a new X.509 extension, DelegationUsage, to be used in the certificate when the certificate permits the usage of delegated credentials. What follows is the ASN.1 [X.680] for the DelegationUsage certificate extension.

```
ext-delegationUsage EXTENSION  ::= {
    SYNTAX DelegationUsage IDENTIFIED BY id-pe-delegationUsage
}

DelegationUsage ::= NULL

id-pe-delegationUsage OBJECT IDENTIFIER ::=
    { iso(1) identified-organization(3) dod(6) internet(1)
      private(4) enterprise(1) id-cloudflare(44363) 44 }
```

The extension MUST be marked non-critical. (See Section 4.2 of [RFC5280].) The client MUST NOT accept a delegated credential unless the server’s end-entity certificate satisfies the following criteria:

* It has the DelegationUsage extension.

* It has the digitalSignature KeyUsage (see the KeyUsage extension defined in [RFC5280]).

A new extension was chosen instead of adding a new Extended Key Usage (EKU) to be compatible with deployed TLS and PKI software stacks without requiring CAs to issue new intermediate certificates.

5. Operational Considerations

5.1. Client Clock Skew

One of the risks of deploying a short-lived credential system based on absolute time is client clock skew. If a client's clock is sufficiently ahead or behind of the server's clock, then clients will reject delegated credentials that are valid from the server's perspective. Clock skew also affects the validity of the original certificates. The lifetime of the delegated credential should be set taking clock skew into account. Clock skew may affect a delegated credential at the beginning and end of its validity periods, which should also be taken into account.

6. IANA Considerations

This document registers the "delegated_credentials" extension in the "TLS ExtensionType Values" registry. The "delegated_credentials" extension has been assigned a code point of 34. The IANA registry lists this extension as "Recommended" (i.e., "Y") and indicates that it may appear in the ClientHello (CH), CertificateRequest (CR), or Certificate (CT) messages in TLS 1.3 [RFC8446].

This document also defines an ASN.1 module for the DelegationUsage certificate extension in Appendix A. IANA has registered value 95 for "id-mod-delegated-credential-extn" in the "SMI Security for PKIX Module Identifier" (1.3.5.1.5.5.7.0) registry. An OID for the DelegationUsage certificate extension is not needed as it is already assigned to the extension from Cloudflare's IANA Private Enterprise Number (PEN) arc.

7. Security Considerations

7.1. Security of Delegated Credential's Private Key

Delegated credentials limit the exposure of the private key used in a TLS connection by limiting its validity period. An attacker who compromises the private key of a delegated credential can act as a man-in-the-middle until the delegated credential expires. However, they cannot create new delegated credentials. Thus, delegated credentials should not be used to send a delegation to an untrusted party, but is meant to be used between parties that have some trust relationship with each other. The secrecy of the delegated credential's private key is thus important and access control mechanisms SHOULD be used to protect it, including file system controls, physical security, or hardware security modules.

7.2. Re-use of Delegated Credentials in Multiple Contexts

It is not possible to use the same delegated credential for both client and server authentication because issuing parties compute the corresponding signature using a context string unique to the intended role (client or server).

7.3. Revocation of Delegated Credentials

Delegated credentials do not provide any additional form of early revocation. Since it is short lived, the expiry of the delegated credential revokes the credential. Revocation of the long term private key that signs the delegated credential (from the end-entity certificate) also implicitly revokes the delegated credential.

7.4. Interactions with Session Resumption

If a client decides to cache the certificate chain and re-validate it when resuming a connection, the client SHOULD also cache the associated delegated credential and re-validate it.

7.5. Privacy Considerations

Delegated credentials can be valid for 7 days and it is much easier for a service to create delegated credentials than a certificate signed by a CA. A service could determine the client time and clock skew by creating several delegated credentials with different expiry timestamps and observing whether the client would accept it. Client time could be unique and thus privacy sensitive clients, such as browsers in incognito mode, who do not trust the service might not want to advertise support for delegated credentials or limit the number of probes that a server can perform.

7.6. The Impact of Signature Forgery Attacks

When TLS 1.2 servers support RSA key exchange, they may be vulnerable to attacks that allow forging an RSA signature over an arbitrary message [BLEI]. TLS 1.2 [RFC5246] (Section 7.4.7.1.) describes a mitigation strategy requiring careful implementation of timing resistant countermeasures for preventing these attacks. Experience shows that in practice, server implementations may fail to fully stop these attacks due to the complexity of this mitigation [ROBOT]. For TLS 1.2 servers that support RSA key exchange using a DC-enabled end-entity certificate, a hypothetical signature forgery attack would allow forging a signature over a delegated credential. The forged delegated credential could then be used by the attacker as the equivalent of a man-in-the-middle certificate, valid for a maximum of 7 days.

Server operators should therefore minimize the risk of using DC-enabled end-entity certificates where a signature forgery oracle may be present. If possible, server operators may choose to use DC-enabled certificates only for signing credentials, and not for serving non-DC TLS traffic. Furthermore, server operators may use elliptic curve certificates for DC-enabled traffic, while using RSA certificates without the DelegationUsage certificate extension for non-DC traffic; this completely prevents such attacks.

Note that if a signature can be forged over an arbitrary credential, the attacker can choose any value for the valid_time field. Repeated signature forgeries therefore allow the attacker to create multiple delegated credentials that can cover the entire validity period of the certificate. Temporary exposure of the key or a signing oracle may allow the attacker to impersonate a server for the lifetime of the certificate.

8. Acknowledgements

Thanks to David Benjamin, Christopher Patton, Kyle Nekritz, Anirudh Ramachandran, Benjamin Kaduk, Kazuho Oku, Daniel Kahn Gillmor, Watson Ladd, Robert Merget, Juraj Somorovsky, Nimrod Aviram for their discussions, ideas, and bugs they have found.

9. References

9.1. Normative References

[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>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[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>.

[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/info/rfc8446>.

[X.680] ITU-T, "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ISO/IEC 8824-1:2015, November 2015.

[X.690] ITU-T, "Information technology - ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ISO/IEC 8825-1:2015, November 2015.

9.2. Informative References

[BLEI] Bleichenbacher, D., "Chosen Ciphertext Attacks against Protocols Based on RSA Encryption Standard PKCS #1", Advances in Cryptology -- CRYPTO'98, LNCS vol. 1462, pages: 1-12 , 1998.

[KEYLESS] Sullivan, N. and D. Stebila, "An Analysis of TLS Handshake Proxying", IEEE Trustcom/BigDataSE/ISPA 2015 , 2015.

[RFC3820] Tuecke, S., Welch, V., Engert, D., Pearlman, L., and M. Thompson, "Internet X.509 Public Key Infrastructure (PKI) Proxy Certificate Profile", RFC 3820, DOI 10.17487/RFC3820, June 2004, <https://www.rfc-editor.org/info/rfc3820>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <https://www.rfc-editor.org/info/rfc5912>.

[RFC8032] Josefsson, S. and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)", RFC 8032, DOI 10.17487/RFC8032, January 2017, <https://www.rfc-editor.org/info/rfc8032>.

[RFC8555] Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", RFC 8555, DOI 10.17487/RFC8555, March 2019, <https://www.rfc-editor.org/info/rfc8555>.

[ROBOT] Boeck, H., Somorovsky, J., and C. Young, "Return Of Bleichenbacher's Oracle Threat (ROBOT)", 27th USENIX Security Symposium , 2018.

[XPROT] Jager, T., Schwenk, J., and J. Somorovsky, "On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 v1.5 Encryption", Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security , 2015.

Appendix A. ASN.1 Module

The following ASN.1 module provides the complete definition of the DelegationUsage certificate extension. The ASN.1 module makes imports from [RFC5912].

```
DelegatedCredentialExtn
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-delegated-credential-extn(TBD) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

-- EXPORT ALL

IMPORTS

EXTENSION
  FROM PKIX-CommonTypes-2009 -- From RFC 5912
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-pkixCommon-02(57) } ;

-- OID

id-cloudflare OBJECT IDENTIFIER ::=
  { iso(1) identified-organization(3) dod(6) internet(1) private(4)
    enterprise(1) 44363 }

-- EXTENSION

ext-delegationUsage EXTENSION ::=
  { SYNTAX DelegationUsage
    IDENTIFIED BY id-pe-delegationUsage }

id-pe-delegationUsage OBJECT IDENTIFIER ::= { id-cloudflare 44 }

DelegationUsage ::= NULL

END
```

Appendix B. Example Certificate

The following certificate has the Delegated Credentials OID.

```
-----BEGIN CERTIFICATE-----
MIIFRjCCBMugAwIBAgIQDGevB+lY0o/OecHFSJ6YnTAKBggqhkjOPQQDAzBMMQsw
CQYDVQQGEwJVUzEVMBMGA1UEChMMRGlnaUNlcnQgSW5jMSYwJAYDVQQDEx1EaWdp
Q2VydCBFQ0MgU2VjdXJlIFNlcnZlciBDQTAeFw0xOTAzMjYwMDAwMDBaFw0yMTAz
MzAxMjAwMDBaMGoxCzAJBgNVBAYTAlVTMRMwEQYDVQQIEwpDYWxpZm9ybmlhMRYw
FAYDVQQHEw1TYW4gRnJhbmNpc2NvMRkwFwYDVQQKExBDbG91ZGZsYXJlLCBJbmMu
MRMwEQYDVQQDEwprYzJrZG0uY29tMFkwEwYHKoZIzj0CAQYIKoZIzj0DAQcDQgAE
d4azI83Bw0fcPgfoeiZpZZnwGuxjBjv++wzE0zAj8vNiUkKxOWSQiGNLn+xlWUpL
lw9djRN1rLmVmn2gb9GgdKOCA28wggNrMB8GA1UdIwQYMBaAFKOd5h/52jlPwG7o
kcuVpdox4gqfMB0GA1UdDgQWBBSfcb7fS3fUFAyB91fRcwoDPtgtJjAjBgNVHREE
HDAaggprYzJrZG0uY29tggwqLmtjMmtkbS5jb20wDgYDVR0PAQH/BAQDAgeAMB0G
A1UdJQQWMBQGCCsGAQUFBwMBBggrBgEFBQcDAjBpBgNVHR8EYjBgMC6gLKAqhiho
dHRwOi8vY3JsMy5kaWdpY2VydC5jb20vc3NjYS1lY2MtZzEuY3JsMC6gLKAqhiho
dHRwOi8vY3JsNC5kaWdpY2VydC5jb20vc3NjYS1lY2MtZzEuY3JsMEwGA1UdIARF
MEMwNwYJYIZIAYb9bAEBMCowKAYIKwYBBQUHAgEWHGh0dHBzOi8vd3d3LmRpZ2lj
ZXJ0LmNvbS9DUFMwCAYGZ4EMAQICMHsGCCsGAQUFBwEBBG8wbTAkBggrBgEFBQcw
AYYYaHR0cDovL29jc3AuZGlnaWNlcnQuY29tMEUGCCsGAQUFBzAChjlodHRwOi8v
Y2FjZXJ0cy5kaWdpY2VydC5jb20vRGlnaUNlcnRFQ0NTZWN1cmVTZXJ2ZXJDQS5j
cnQwDAYDVR0TAQH/BAIwADAPBgkrBgEEAYLaSywEAgUAMIIBfgYKKwYBBAHWeQIE
AgSCAW4EggFqAWgAdgC72d+8H4pxtZOUI5eqkntHOFeVCqtS6BqQlmQ2jh7RhQAA
AWm5hYJ5AAAEAwBHMEUCICiGfq+hSThRL2m8H0awoDR8OpnEHNkF0nI6nL5yYL/j
AiEAxwebGs/T6Es0YarPzoQJrVZqk+sHH/t+jrSrKd5TDjcAdgCHdb/nWXz4jEOZ
X73zbv9WjUdWNv9KtWDBtOr/XqCDDwAAAWm5hYNgAAAEAwBHMEUCIQD9OWA8KGL6
bxDKfgIleHJWB0iWieRs88VgJyfAg/aFDgIgQ/OsdSF9XOy1foqge0DTDM2FExuw
0JR0AGZWXoNtJzMAdgBElGUusO7Or8RAB9io/ijA2uaCvtjLMbU/0zOWtbaBqAAA
AWm5hYHgAAAEAwBHMEUCIQC4vua1n3BqthEqpA/VBTcsNwMtAwpCuac2IhJ9wx6X
/AIgb+o00k28JQo9TMpP4vzJ3BD3HXWSNc2Zizbq7mkUQYMwCgYIKoZIzj0EAwMD
aQAwZgIxAJsX7d0SuA8ddf/m7IWfNfs3MQfJyGkEezMJX1t6sRso5z50SS12LpXe
muGa1FE2ZgIxAL+CDUF5pz7mhrAEIjQ1MqlpF9tH40dJGvYZZQ3W23cMzSkDfvlt
y5S4RfWHIIPjbw==
-----END CERTIFICATE-----
```

Authors' Addresses

Richard Barnes
Cisco

Email: rlb@ipv.sx

Subodh Iyengar
Facebook

Email: subodh@fb.com

Nick Sullivan
Cloudflare

Email: nick@cloudflare.com

Eric Rescorla
Mozilla

Email: ekr@rtfm.com

TLS   Y. Nir
Internet-Draft   Dell Technologies
Intended status: Standards Track   July 3, 2020
Expires: January 4, 2021

A Flags Extension for TLS 1.3
draft-ietf-tls-tlsflags-03

Abstract

A number of extensions are proposed in the TLS working group that carry no interesting information except the 1-bit indication that a certain optional feature is supported. Such extensions take 4 octets each. This document defines a flags extension that can provide such indications at an average marginal cost of 1 bit each. More precisely, it provides as many flag extensions as needed at 4 + the order of the last set bit divided by 8.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Since the publication of TLS 1.3 ([RFC8446]) there have been several proposals for extensions to this protocol, where the presence of the content-free extension in both the ClientHello and either the ServerHello or EncryptedExtensions indicates nothing except either support for the optional feature or an intent to use the optional feature. Examples:

- An extension that allows the server to tell the client that cross-SNI resumption is allowed: [I-D.sy-tls-resumption-group].

- An extension that is used to negotiate support for authentication using both certificates and external PSKs: [I-D.ietf-tls-tls13-cert-with-extern-psk].

- The post_handshake_auth extension from the TLS 1.3 base document indicates that the client is willing to perform post-handshake authentication.

This document proposes a single extension called tls_flags that can enumerate such flag extensions and allowing both client and server to indicate support for optional features in a concise way.

None of the current proposed extensions are such that the server indicates support without the client first indicating support. This specification enforces this restriction by specifying in Section 3 that server bits may only reflect flags for which the client extension has already indicated support.

1.1. Requirements and Other Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD 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.

The term "flag extension" is used to denote an extension where the extension_data field is always zero-length in a particular context, and the presence of the extension denotes either support for some feature or the intent to use that feature.

The term "flag-type feature" denotes an options TLS 1.3 feature the support for which is negotiated using a flag extension, whether that flag extension is its own extension or a value in the extension defined in this document.

2. The tls_flags Extension

This document defines the following extension code point:

```
enum {
   ...
   tls_flags(TBD),
   (65535)
} ExtensionType;
```

This document also defines the data for this extension as a variable-length bit string, allowing for the encoding of up to 2040 features.

```
struct {
   opaque flags<0..255>;
} FlagExtensions;
```

The FlagExtensions field 8 flags with each octet, and its length is the minimal length that allows it to encode all of the present flags. Within each octet, the bits are packed such that the first bit is the LSB and the seventh bit is the MSB. The first octet holds flags 0-7, the second octet holds bits 8-15 and so on. For example, if we want to encode only flag number zero, the FlagExtension field will be 1 octet long, that is encoded as follows:

```
00000001
```

If we want to encode flags 1 and 5, the field will still be 1 octet long:

```
00100010
```

If we want to encode flags 3, 5, and 23, the field will have to be 3 octets long:

```
00101000 00000000 10000000
```

An implementation that receives an all-zero value for this extension or a value that contains trailing zero bytes MUST generate a fatal illegal_parameter alert.

Note that this document does not define any particular bits for this string. That is left to the protocol documents such as the ones in the examples from the previous section. Such documents will have to define which bit to set to show support, and the order of the bits within the bit string shall be enumerated in network order: bit zero is the high-order bit of the first octet as the flags field is transmitted.

3. Rules for The Flags Extension

A client that supports this extension and at least one flag extension SHALL send this extension with the flags field having bits set only for those extensions that it intends to set. It MUST NOT send this extension with a length of zero.

A server that supports this extension and also supports at least one of the flag-type features that use this extension and that were declared by the ClientHello extension SHALL send this extension with the intersection of the flags it supports with the flags declared by the client. The intersection operation MAY be implemented as a bitwise AND. The server may need to send two tls_flags extensions, one in the ServerHello and the other in the EncryptedExtensions message. It is up to the document for the specific feature to determine whether support should be acknowledged in the ServerHello or the EncryptedExtensions message.

A server MUST NOT indicate support for any flag-type feature not previously indicated by the client. It MUST NOT include this extension in either message (ServerHello or EncryptedExtensions) if it has no appropriate flag-type to indicate. This extension MUST NOT be included empty.

4. IANA Considerations

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

- The Extension Name should be tls_flags

- The TLS 1.3 value should be CH,SH,EE

- The Recommended value should be Y

- The Reference should be this document

IANA is also requested to create a new registry under the TLS namespace with name "TLS Flags" and the following fields:

- Value, which is a number between 0 and 2039. All potential values are available for assignment.

- Flag Name, which is a string

- Message, which like the "TLS 1.3" field in the ExtensionType registry contains the abbreviations of the messages that may contain the flag: CH, SH, EE, etc.

- Recommended, which is a Y/N value determined in the document defining the optional feature.

- Reference, which is a link to the document defining this flag.

The policy for this shall be "Specification Required" as described in [RFC8126].

4.1. Guidance for IANA Experts

This extension allows up to 2040 flags. However, they are not all the same, because the length of the extension is determined by the highest set bit.

We would like to allocate the flags in such a way that the typical extension is as short as possible. The scenario we want to guard against is that in a few years some extension is defined that all implementations need to support and that is assigned a high number because all of the lower numbers have already been allocated. An example of such an extension is the Renegotiation Indication Extension defined in [RFC5746].

For this reason, the IANA experts should allocate the flags as follows:

- Flags 0-7 are reserved for documents coming out of the TLS working group with a specific request to assign a low number.

- Flags 8-31 are for standards-track documents that the experts believe will see wide adoption among either all users of TLS or a significant group of TLS users. For example, an extension that will be used by all web clients or all smart objects.

- Flags 32-63 are for other documents, including experimental, that are likely to see significant adoption.

- Flags 64-79 are not to be allocated. They are for reserved for private use.

- Flags 80-2039 can be used for temporary allocation in experiments, for flags that are likely to see use only in very specific environments, for national and corporate extensions, and as overflow, in case one of the previous categories has been exhausted.

5. Security Considerations

The extension described in this document provides a more concise way to express data that could otherwise be expressed in individual extensions. It does not send in the clear any information that would otherwise be sent encrypted, nor vice versa. For this reason this extension is neutral as far as security is concerned.

6. Acknowledgements

The idea for writing this was expressed at the mic during the TLS session at IETF 104 by Eric Rescorla.

The current bitwise formatting was suggested on the mailing list by Nikos Mavrogiannopoulos.

Improvement to the encoding were suggested by Ilari Liusvaara, who also asked for a better explanation of the semantics of missing extensions.

Useful comments received from Martin Thomson, including the suggestion to eliminate the option to have the server send unsolicited flag types.

7. References

7.1. Normative References

[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>.

[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/info/rfc8446>.

7.2. Informative References

[I-D.ietf-tls-tls13-cert-with-extern-psk]
Housley, R., "TLS 1.3 Extension for Certificate-based Authentication with an External Pre-Shared Key", draft-ietf-tls-tls13-cert-with-extern-psk-07 (work in progress), December 2019.

[I-D.sy-tls-resumption-group]
Sy, E., "TLS Resumption across Server Name Indications for TLS 1.3", draft-sy-tls-resumption-group-00 (work in progress), March 2019.

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <https://www.rfc-editor.org/info/rfc5746>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

Appendix A. Change Log

RFC EDITOR: PLEASE REMOVE THIS SECTION AS IT IS ONLY MEANT TO AID THE WORKING GROUP IN TRACKING CHANGES TO THIS DOCUMENT.

draft-ietf-tls-tlsflags-02 set the maximum number of flags to 2048, and added guidance for the IANA experts.

draft-ietf-tls-tlsflags-01 allows server-only flags and allows the client to send an empty extension. Also modified the packing order of the bits.

draft-ietf-tls-tlsflags-00 had the same text as draft-nir-tls-tlsflags-02, and was re-submitted as a working group document following the adoption call.

Version -02 replaced the fixed 64-bit string with an unlimited bitstring, where only the necessary octets are encoded.

Version -01 replaced the enumeration of 8-bit values with a 64-bit bitstring.

Version -00 was a quickly-thrown-together draft with the list of supported features encoded as an array of 8-bit values.

Author's Address

Yoav Nir
Dell Technologies
9 Andrei Sakharov St
Haifa 3190500
Israel

Email: ynir.ietf@gmail.com

jhoyla J. Hoyland
Internet-Draft Cloudflare Ltd.
Intended status: Standards Track C.A. Wood
Expires: 10 September 2020 Apple, Inc.
9 March 2020

TLS 1.3 Extended Key Schedule
draft-jhoyla-tls-extended-key-schedule-01

Abstract

TLS 1.3 is sometimes used in situations where it is necessary to inject extra key material into the handshake. This draft aims to describe methods for doing so securely. This key material must be injected in such a way that both parties agree on what is being injected and why, and further, in what order.

Note to Readers

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/ (https://mailarchive.ietf.org/arch/browse/tls/).

Source for this draft and an issue tracker can be found at https://github.com/jhoyla/draft-jhoyla-tls-key-injection (https://github.com/jhoyla/draft-jhoyla-tls-key-injection).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Introducing additional key material into the TLS handshake is a non-trivial process because both parties need to agree on the injection content and context. If the two parties do not agree then an attacker may exploit the mismatch in so-called channel synchronization attacks.

Injecting key material into the TLS handshake allows other protocols to be bound to the handshake. For example, it may provide additional protections to the ClientHello message, which in the standard TLS handshake only receives protections after the server's Finished message has been received. It may also permit the use of combined shared secrets, possibly from multiple key exchange algorithms, to be included in the key schedule. This pattern is common for Post Quantum key exchange algorithms, as discussed in [I-D.stebila-tls-hybrid-design].

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. Key Schedule Extension

This section describes two places in which additional secrets can be injected into the TLS 1.3 key schedule.

3.1. Handshake Secret Injection

To inject key material into the Handshake Secret it is recommended to use an extra derive secret.

```
                 |
                 v
          Derive-Secret(., "derived early", "")
                 |
                 v
       Input -> HKDF-Extract
                 |
                 v
          Derive-Secret(., "derived", "")
                 |
                 v
     (EC)DHE -> HKDF-Extract = Handshake Secret
                 |
                 v
```

As shown in the figure above, the key schedule has an extra derive secret and HKDF-Extract step. This extra step isolates the Input material from the rest of the handshake secret, such that even maliciously chosen values cannot weaken the security of the key schedule overall.

The additional Derive-Secret with the "derived early" label enforces the separation of the key schedule from vanilla TLS handshakes, because HKDFs can be assumed to ensure that keys derived with different labels are independent.

3.2. Master Secret Injection

To inject key material into the Master Secret it is recommended to use an extra derive secret.

```
                |
                v
          Derive-Secret(., "derived early", "")
                |
                v
       Input -> HKDF-Extract
                |
                v
          Derive-Secret(., "derived", "")
                |
                v
       0 -> HKDF-Extract = Master Secret
                |
                v
```

This structrue mirrors the Handshake Injection point, the key schedule has an extra Extract, Derive-Secret pattern. This, again, should isolate the Input material from the rest of the Master Secret.

4. Key Schedule Extension Structure

In some cases, protocols may require more than one secret to be injected at a particular stage in the key schedule. Thus, we require a generic and extensible way of doing so. To accomplish this, we use a structure - KeyScheduleInput - that encodes well-ordered sequences of secret material to inject into the key schedule. KeyScheduleInput is defined as follows:

```
struct {
    KeyScheduleSecretType type;
    opaque secret_data<0..2^16-1>;
} KeyScheduleSecret;

enum {
    (65535)
} KeyScheduleSecretType;

struct {
    KeyScheduleSecret secrets<0..2^16-1>;
} KeyScheduleInput;
```

Each secret included in a KeyScheduleInput structure has a type and corresponding secret data. Each secret MUST have a unique KeyScheduleSecretType. When encoding KeyScheduleInput as the key schedule Input value, the KeyScheduleSecret values MUST be in ascending sorted order. This ensures that endpoints always encode the same KeyScheduleInput value when using the same secret keying material.

5. Security Considerations

[[OPEN ISSUE: This draft has not seen any security analysis.]]

6. IANA Considerations

[[TODO: define secret registry structure]]

7. References

7.1. Normative References

[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>.

7.2. Informative References

[I-D.stebila-tls-hybrid-design]
Steblia, D., Fluhrer, S., and S. Gueron, "Hybrid key exchange in TLS 1.3", Work in Progress, Internet-Draft, draft-stebila-tls-hybrid-design-03, 12 February 2020, <http://www.ietf.org/internet-drafts/draft-stebila-tls-hybrid-design-03.txt>.

Acknowledgments

We thank Karthik Bhargavan for his comments.

Authors' Addresses

Jonathan Hoyland
Cloudflare Ltd.

Email: jonathan.hoyland@gmail.com

Christopher A. Wood
Apple, Inc.

Email: cawood@apple.com

jhoyla   J. Hoyland
Internet-Draft   Cloudflare Ltd.
Intended status: Standards Track   C.A. Wood
Expires: 7 June 2021   Cloudflare
                                                          4 December 2020

TLS 1.3 Extended Key Schedule
draft-jhoyla-tls-extended-key-schedule-03

Abstract

TLS 1.3 is sometimes used in situations where it is necessary to inject extra key material into the handshake. This draft aims to describe methods for doing so securely. This key material must be injected in such a way that both parties agree on what is being injected and why, and further, in what order.

Note to Readers

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/ (https://mailarchive.ietf.org/arch/browse/tls/).

Source for this draft and an issue tracker can be found at https://github.com/jhoyla/draft-jhoyla-tls-key-injection (https://github.com/jhoyla/draft-jhoyla-tls-key-injection).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Introducing additional key material into the TLS handshake is a non-trivial process because both parties need to agree on the injection content and context. If the two parties do not agree then an attacker may exploit the mismatch in so-called channel synchronization attacks, such as those described by [SLOTH].

Injecting key material into the TLS handshake allows other protocols to be bound to the handshake. For example, it may provide additional protections to the ClientHello message, which in the standard TLS handshake only receives protections after the server's Finished message has been received. It may also permit the use of combined shared secrets, possibly from multiple key exchange algorithms, to be included in the key schedule. This pattern is common for Post Quantum key exchange algorithms, as discussed in

[I-D.ietf-tls-hybrid-design]. In particular, [I-D.ietf-tls-hybrid-design] uses the concatenation pattern described in this draft, but does not add the requisite framing.

The goal of this document is to provide a standardised way for binding extra context into TLS 1.3 handshakes in a way that is easy to analyse from a security perspective, reducing the need for security analysis of extensions that affect the key schedule. It separates the concerns of whether an extension achieves its goals from the concerns of whether an extension reduces the security of a TLS handshake, either directly or through some unforseen interaction with another extension.

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. Key Schedule Extension

This section describes two places in which additional secrets can be injected into the TLS 1.3 key schedule.

3.1. Handshake Secret Injection

To inject extra key material into the Handshake Secret it is recommended to prefix it, inside an appropriate frame, to the "(EC)DHE" input, where "||" represents concatenation.

```
                                    |
                                    v
                              Derive-Secret(., "derived", "")
                                    |
                                    v
     KeyScheduleInput || (EC)DHE -> HKDF-Extract = Handshake Secret
                                    |
                                    v
```

3.2. Main Secret Injection

To inject key material into the Main Secret it is recommended to prefix it, inside an appropriate frame, to the "0" input.

```
                              |
                              v
                        Derive-Secret(., "derived", "")
                              |
                              v
     KeyScheduleInput || 0 -> HKDF-Extract = Main Secret
                              |
                              v
```

This structure mirrors the Handshake Injection point.

4. Key Schedule Injection Negotiation

Applications which make use of additional key schedule inputs MUST define a mechanism for negotiating the content and type of that input. This input MUST be framed in a KeyScheduleSecret struct, as defined in Section 5. Applications must take care that any negotiation that takes place unambiguously agrees a secret. It must be impossible, even under adversarial conditions, that a client and server agree on the transcript of the negotiation, but disagree on the secret that was negotiated.

5. Key Schedule Extension Structure

In some cases, protocols may require more than one secret to be injected at a particular stage in the key schedule. Thus, we require a generic and extensible way of doing so. To accomplish this, we use a structure - KeyScheduleInput - that encodes well-ordered sequences of secret material to inject into the key schedule. KeyScheduleInput is defined as follows:

```
struct {
    KeyScheduleSecretType type;
    opaque secret_data<0..2^16-1>;
} KeyScheduleSecret;

enum {
    (65535)
} KeyScheduleSecretType;

struct {
    KeyScheduleSecret secrets<0..2^16-1>;
} KeyScheduleInput;
```

Each secret included in a KeyScheduleInput structure has a type and corresponding secret data. Each secret MUST have a unique KeyScheduleSecretType. When encoding KeyScheduleInput as the key schedule Input value, the KeyScheduleSecret values MUST be in ascending sorted order. This ensures that endpoints always encode the same KeyScheduleInput value when using the same secret keying material.

6. Security Considerations

[BINDEL] provides a proof that the concatenation approach in Section 3 is secure as long as either the concatenated secret is secure or the existing KDF input is secure.

[[OPEN ISSUE: Is this guarantee sufficient? Do we also need to guarantee that a malicious prefix can’t weaken the resulting PRF output?]]

7. IANA Considerations

This document requests the creation of a new IANA registry: TLS KeyScheduleInput Types. This registry should be under the existing Transport Layer Security (TLS) Parameters heading. It should be administered under a Specification Required policy [RFC8126].

[[OPEN ISSUE: specify initial registry values]]

Value	Description	DTLS-OK	Reference
TBD	TBD	TBD	TBD

Table 1

8. References

8.1. Normative References

[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>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[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>.

8.2. Informative References

[BINDEL] Bindel, N., Brendel, J., Fischlin, M., Goncalves, B., and D. Stebila, "Hybrid Key Encapsulation Mechanisms and Authenticated Key Exchange", Post-Quantum Cryptography pp. 206-226, DOI 10.1007/978-3-030-25510-7_12, 2019, <https://doi.org/10.1007/978-3-030-25510-7_12>.

[I-D.friel-tls-eap-dpp]
Friel, O. and D. Harkins, "Bootstrapped TLS Authentication", Work in Progress, Internet-Draft, draft-friel-tls-eap-dpp-01, 13 July 2020, <http://www.ietf.org/internet-drafts/draft-friel-tls-eap-dpp-01.txt>.

[I-D.ietf-tls-hybrid-design]
Steblia, D., Fluhrer, S., and S. Gueron, "Hybrid key exchange in TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-hybrid-design-01, 15 October 2020, <http://www.ietf.org/internet-drafts/draft-ietf-tls-hybrid-design-01.txt>.

[I-D.ietf-tls-semistatic-dh]
Rescorla, E., Sullivan, N., and C. Wood, "Semi-Static Diffie-Hellman Key Establishment for TLS 1.3", Work in Progress, Internet-Draft, draft-ietf-tls-semistatic-dh-01, 7 March 2020, <http://www.ietf.org/internet-drafts/draft-ietf-tls-semistatic-dh-01.txt>.

[SLOTH] Bhargavan, K. and G. Leurent, "Transcript Collision Attacks: Breaking Authentication in TLS, IKE, and SSH", Proceedings 2016 Network and Distributed System Security Symposium, DOI 10.14722/ndss.2016.23418, 2016, <https://doi.org/10.14722/ndss.2016.23418>.

Appendix A. Potential Use Cases

The draft provides a mechanism for importing additional information into the TLS key schedule. Future applications and specifications can use this mechanism to layer TLS on to other protocols, as opposed to layering other protocols over TLS. For example, as discussed in Section 1, this can be used for hybrid key exchange, which, in effect, is layering TLS over a secondary AKE. Although the key exchanges are interleaved, the post-quantum AKE completes first, as demonstrated by its output key being used as an input for computing TLS's master secret.

This can also be used in more direct ways, such as bootstrapping EAP-TLS as in [I-D.friel-tls-eap-dpp]. This draft also allows for more direct implementations of things such as semi-static DH [I-D.ietf-tls-semistatic-dh]. The aim of this draft is to be sufficiently flexible that it can be used as the basis for layering TLS on top of any protocol that outputs a secure channel binding, where secure is defined by the goals of the overall layered protocol. This draft does not provide security itself, it simply provides a standard format for layering.

Acknowledgments

We thank Karthik Bhargavan for his comments.

Authors' Addresses

Jonathan Hoyland
Cloudflare Ltd.

Email: jonathan.hoyland@gmail.com

Christopher A. Wood
Cloudflare

Email: caw@heapingbits.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 14 January 2021

M. Thomson
Mozilla

13 July 2020

Secure Negotiation of Incompatible Protocols in TLS
draft-thomson-tls-snip-00

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, 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

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 for 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-prefered, mutually-incompatible options are available. The client only offers compatible options to a server in its 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.

4. Authenticating Incompatible Protocols

The incompatible_protocols(TBD) TLS extension provides clients with information about the incompatible protocols that are supported by servers.

```
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 "ProtocolNameList" format defined in [ALPN]. This syntax is shown in Figure 1.

```
struct {
  select (Handshake.msg_type) {
    case client_hello:
      Empty;
    case encrypted_extensions:
      ProtocolNameList incompatible_protocols;
  };
} 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. A server MAY limit this to protocols that it considers to have similar semantics to protocols that the client lists in its application_layer_protocol_negotiation extension.

The definition of what a server includes is intentionally loose. It is better that a server offer more information than less as the needs of a client are not necessarily well reflected in its ALPN extension. However, it is not reasonable to require that a server advertise all potential protocols as that is unlikely to be practical.

A server MUST omit any compatible protocols from this extension on the understanding that the client will include compatible protocols in the application_layer_protocol_negotiation extension.

A server needs to ensure that protocols advertised in this fashion are available to the client within the same protocol authentication scope.

5. Protocol Authentication Scope

The protocol authentication scope is the set of protocol endpoints at a server that share a protocol configuration. A client learns of this scope as part of the process it follows to discover the server.

By default, the protocol authentication scope is a single protocol endpoint. The default protocol authentication scope offers no means to authenticate incompatible protocols as it is not possible for a client to access any endpoint that supports those protocols. A client cannot use information from the incompatible_protocols extension unless a wider scope is used.

[[TODO: This likely needs some discussion.]]

5.1. SVCB Discovery Scope

For SVCB records, the protocol authentication scope is defined by the set of ServiceForm SVCB records with the same SvcDomainName.

This ensures that the final choice a client makes between ServiceForm SVCB records is protected by this extension. If the client does not receive a SVCB record for a protocol that the server includes in its incompatible_protocols extension, then it can assume that this omission was caused by an error or attack.

Thus, for SVCB, a choice between AliasForm records (or CNAME or DNAME records) is not authenticated, but choices between ServiceForm records is. This allows for server deployments for the same name to have different administrative control and protocol configurations.

5.2. QUIC Version Negotiation

TODO: define how this can be used to authenticate protocol choices where there are incompatible QUIC versions.

5.3. 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 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 server that is discovered using Alternative Services uses the default protocol authentication scope. As use of Alternative Services is discretionary for both client and server, a client cannot expect to receive information about incompatible protocols. To avoid downgrade, a client only has to avoid using an Alternative Service that offers a less-preferred protocol.

5.4. Scope for Other Discovery Methods

For other discovery methods, a definition for protocol authentication scope is needed before a client can act on what is learned using the incompatible_protocols extension. That definition needs to define how to discover server instances that support all incompatible protocols in the scope.

In particular, a server that is discovered using forms of DNS-based name resolution other than SVCB uses the default protocol authentication scope. This discovery method does not provide enough information to locate other incompatible protocols.

For instance, an HTTPS server that is discovered using purely A or AAAA records (and CNAME or DNAME records) might advertise support for incompatible protocols, but as there is no way to determine where those protocols are supported, a client cannot act on the information. Note that Alternative Services do not change the protocol authentication scope.

Deployments of discovery methods that define a protocol authentication scope larger than the default need to ensure that every server provides information that is consistent with every protocol authentication scope that includes that server. A server that fails to indicate support for a protocol that is within a protocol authentication scope does not offer any protection against attack; a server that advertises a protocol that the client cannot discover risks this misconfiguration being identified as an attack by clients.

6. Incompatible Protocol Selection

This represents a different model for protocol selection than the one used by ALPN. In ALPN, the client presents a set of (compatible) options and the server chooses its most preferred.

In comparison, as the client makes a selection between incompatible protocols before making a connection attempt, this design only provides the client with information about other incompatible protocols that the server might support. Any choice to attempt a connection using those protocols is left to the client.

In summary:

* For compatible protocols, the server chooses

* For incompatible protocols, the client chooses

Detecting a potential downgrade between incompatible protocols does not automatically imply that a client abandon a connection attempt. This 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 the server prefers that protocol. 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 response when a server that indicates HTTP/3 support is unreachable.

7. 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.

A server deployment that uses AliasForm SVCB records and does not uniformly support a client-preferred protocol is vulnerable to downgrade attacks that steer clients toward instances that lack support for that protocol. This attack is ineffective for protocols that are consistently supported by all server instances.

8. IANA Considerations

TODO: register the extension

9. References

9.1. 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/info/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/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>.

9.2. 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/info/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-dtls13-38, 29 May 2020, <http://www.ietf.org/internet-drafts/draft-ietf-tls-dtls13-38.txt>.

[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/info/rfc6555>.

[HTTP11] Fielding, R., Nottingham, M., and J. Reschke, "HTTP/1.1 Messaging", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-10, 12 July 2020, <http://www.ietf.org/internet-drafts/draft-ietf-httpbis-messaging-10.txt>.

[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/info/rfc7540>.

[HTTP3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-http-29.txt>.

[QUIC-TLS] Thomson, M. and S. Turner, "Using TLS to Secure QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-tls-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-29.txt>.

[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/info/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, <http://www.ietf.org/internet-drafts/draft-ietf-dnsop-svcb-httpssvc-03.txt>.

[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>.

Appendix A. Acknowledgments

Author's Address

Martin Thomson
Mozilla

Email: mt@lowentropy.net

Network Working Group
Internet-Draft
Intended status: Informational
Expires: 8 July 2021

M. Thomson
Mozilla

4 January 2021

Secure Negotiation of Incompatible Protocols in TLS
draft-thomson-tls-snip-01

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, 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

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.

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-prefered, mutually-incompatible options are available. The client only offers compatible options to a server in its 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.

4. Authenticating Incompatible Protocols

The incompatible_protocols(TBD) TLS extension provides clients with information about the incompatible protocols that are supported by servers.

```
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.

```
enum {
  default(0), svcb(1), quic(2), (255)
} ProtocolAuthenticationScope;

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

struct {
  ProtocolAuthenticationScope scope;
  ProtocolName protocol;
} 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. Each protocol is paired with an identifier for the Protocol Authentication Scope, which defines how endpoints for that protocol might be discovered; see Section 6.

A server needs to ensure that protocols advertised in this fashion are available to the client within the same protocol authentication scope.

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. Clients are expected to include all compatible protocols in the application_layer_protocol_negotiation extension.

A server MAY limit the incompatible protocols that it advertises to those that have similar semantics to protocols that the client lists in its application_layer_protocol_negotiation extension.

The definition of what a server includes is intentionally flexible. It is better that a server offer more information than less as the needs of a client are not necessarily well reflected in its ALPN extension. However, it might not be feasible for a server to advertise all potential protocols; see Section 8 for more discussion on this point.

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. Protocol Authentication Scope

A protocol authentication scope includes a set of service endpoints that are provided downgrade protection by this mechanism. There are multiple types of protocol authentication scope, each identified by a different type. The type of protocol authentication scope is encoded in the "ProtocolAuthenticationScope" enum.

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 all of the following occur:

1. A server advertises that there are endpoints that support a protocol that the client prefers over the protocol that is currently in use.

2. The protocol authentication scope associated with that protocol is understood by the client and the client attempted to discover services in that scope.

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.

6.1. The Default Scope

The default protocol authentication scope reserves an identifier of 0. A client cannot act on information about incompatible protocols advertised with this scope. A server MUST NOT advertise incompatible protocols with this scope; however, a client MUST ignore advertisements it receives.

The default protocol authentication scope is reserved for discovery methods that have no explicit scope; see Section 7 for more on this subject.

6.2. SVCB Scope

The SVCB protocol authentication scope uses an identifier of 1. A server that lists incompatible protocols with this scope indicates that SVCB records ServiceForm records with the same SvcDomainName exist that refer to services that support the indicated protocol.

The SVCB protocol authentication scope also applies to records that use the SVCB form, like HTTPS.

This ensures that the final choice a client makes between ServiceForm SVCB records is protected by this extension. If the client does not receive a SVCB record for a protocol that the server includes in its incompatible_protocols extension, then it can assume that this omission was caused by an error or attack.

A choice between AliasForm records (or CNAME or DNAME records) is not authenticated, but choices between ServiceForm records is. This allows for server deployments for the same name to have different administrative control and protocol configurations.

6.3. QUIC Version Negotiation Scope

The QUIC version negotiation protocol authentication scope uses an identifier of 2. A server that lists incompatible protocols with this scope indicates that QUIC version negotiation at the same server IP and port could be used to learn of incompatible QUIC versions that support the indicated protocol.

Using this protocol authentication scope depends on application protocols that are dependent on a specific QUIC version.

7. Other Discovery Methods

For other discovery methods, a definition for protocol authentication scope is needed before a client can act on what is learned using the incompatible_protocols extension. That definition needs to define how to discover server instances that support all incompatible protocols in the scope.

In particular, a server that is discovered using forms of DNS-based name resolution other than SVCB uses the default protocol authentication scope; see Section 6.1. Discovering services in this way does not provide enough information to locate other incompatible protocols.

For instance, an HTTPS server that is discovered using purely A or AAAA records (and CNAME or DNAME records) might advertise support for incompatible protocols, but as there is no way to determine where those protocols are supported, a client cannot act on the information. Note that Alternative Services do not change the protocol authentication scope.

Deployments of discovery methods that define a protocol authentication scope larger than the default need to ensure that every server provides information that is consistent with every protocol authentication scope that includes that server. A server that fails to indicate support for a protocol that is within a protocol authentication scope does not offer any protection against attack; a server that advertises a protocol that the client cannot discover risks this misconfiguration being identified as an attack by clients.

7.1. 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 server that is discovered using Alternative Services uses the default protocol authentication scope. As use of Alternative Services is discretionary for both client and server, a client cannot expect to receive information about incompatible protocols. To avoid downgrade, a client only has to limit its use of Alternative Services to those that it prefers more than the active protocol.

8. Operational Considerations

By listing incompatible protocols, a server does not indicate how to find endpoints that support those protocols, only that they exist. This ensures that server configuration is minimized, as servers do not require tight coordination. Providing even this much information could present operational difficulties as it requires that incompatible protocols are only listed when those protocols are deployed.

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 available, incompatible protocols, clients cannot learn about other services and so cannot detect downgrade attacks against those protocols.

9. 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.

A server deployment that uses AliasForm SVCB records and does not uniformly support a client-preferred protocol is vulnerable to downgrade attacks that steer clients toward instances that lack support for that protocol. This attack is ineffective for protocols that are consistently supported by all server instances.

10. IANA Considerations

TODO: register the extension

TODO: create a registry of scopes

11. References

11.1. 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/info/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/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

[ALTSVC]   Nottingham, M., McManus, P., and J. Reschke, "HTTP Alternative Services", RFC 7838, DOI 10.17487/RFC7838, April 2016, <https://www.rfc-editor.org/info/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-dtls13-39, 2 November 2020, <http://www.ietf.org/internet-drafts/draft-ietf-tls-dtls13-39.txt>.

[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/info/rfc6555>.

[HTTP11] Fielding, R., Nottingham, M., and J. Reschke, "HTTP/1.1", Work in Progress, Internet-Draft, draft-ietf-httpbis-messaging-13, 14 December 2020, <http://www.ietf.org/internet-drafts/draft-ietf-httpbis-messaging-13.txt>.

[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/info/rfc7540>.

[HTTP3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", Work in Progress, Internet-Draft, draft-ietf-quic-http-33, 15 December 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-http-33.txt>.

[QUIC-TLS] Thomson, M. and S. Turner, "Using TLS to Secure QUIC", Work in Progress, Internet-Draft, draft-ietf-quic-tls-33, 13 December 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-33.txt>.

[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/info/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, <http://www.ietf.org/internet-drafts/draft-ietf-dnsop-svcb-httpssvc-03.txt>.

[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>.

Appendix A. Acknowledgments

Benjamin Schwartz provided significant input into the design of the mechanism and helped clarify many points.

Author's Address

Martin Thomson
Mozilla

Email: mt@lowentropy.net

TLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 28 December 2020

V. Vasiliev
Google

26 June 2020

TLS Application-Layer Protocol Settings Extension

draft-vvv-tls-alps-00

Abstract

This document describes a Transport Layer Security (TLS) extension for negotiating application-layer protocol settings (ALPS) within the TLS handshake. Any application-layer protocol operating over TLS can use this mechanism to indicate its settings to the peer in parallel with the TLS handshake completion.

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/ (https://mailarchive.ietf.org/arch/browse/tls/).

Source for this draft and an issue tracker can be found at https://github.com/vasilvv/tls-alps (https://github.com/vasilvv/tls-alps).

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 28 December 2020.

Copyright Notice

Table of Contents

1. Introduction

An application-layer protocol often starts with both parties negotiating parameters under which the protocol operates; for instance, HTTP/2 [RFC7540] uses a SETTINGS frame to exchange the list of protocol parameters supported by each endpoint. This is usually achieved by waiting for TLS handshake [RFC8446] to complete and then performing the application-layer handshake within the application protocol itself. This approach, despite its apparent simplicity at first, has multiple drawbacks:

1. While the server is technically capable of sending configuration to the peer as soon as it sends its Finished message, most TLS implementations do not allow any application data to be sent until the Finished message is received from the client. This adds an extra round-trip to the time of when the server settings are available to the client.

2. In QUIC, any settings delivered within the application layer can arrive after other application data; thus, the application has to operate under the assumption that peer's settings are not always available.

3. If the application needs to be aware of the server settings in order to send 0-RTT data, the application has to manually integrate with the TLS stack to associate the settings with TLS session tickets.

This document introduces a new TLS extension, "application_settings", that allows applications to exchange settings within the TLS handshake. Through doing that, the settings can be made available to the application as soon as the handshake completes, and can be associated with TLS session tickets automatically at the TLS layer. This approach allows the application protocol to be designed with the assumption that it has access to the peer's settings whenever it is able to send data.

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. Semantics

Settings are defined to be an opaque blob that is specified by the application when initiating a TLS connection. The settings are meant to be a _declaration_ of the protocol parameters supported by the sender. While in this version of the extension the server settings are always sent first, this may change in future versions; thus, the application MUST NOT vary client settings based on the ones received from the server.

ALPS is _not_ a negotiation mechanism: there is no notion of rejecting peer's settings, and the settings are not responses to one another. Nevertheless, it is possible for parties to coordinate behavior by, for instance, requiring a certain parameter to be present in both client and server settings. This makes ALPS mechanism similar to QUIC transport parameters [I-D.ietf-quic-transport] or HTTP/2 SETTINGS frame [RFC7540], but puts it in contrast to similar mechanisms in TLS.

Settings are exchanged as a part of the TLS handshake that is encrypted with the handshake keys. When the server settings are sent, the identity of the client has not been yet established; therefore, an application MUST NOT use ALPS if it requires the settings to be available only to the authenticated clients.

The ALPS model provides applications with a guarantee that the settings are available before any application data can be written. Note that this implies that when the full handshake is performed, the server can no longer send data immediately after sending its Finished message; it has to wait for the client to respond with its settings. This may negatively impact the latency of the protocols where the server sends the first message, however it should be noted that sending application data before receiving has not been widely supported by TLS implementations, nor has it been allowed in situations when establishing client identity through TLS is required.

ALPS can only be used in conjunction with Application-Layer Protocol Negotiation: the client MUST offer ALPN [RFC7301] if advertising ALPS support, and the server MUST NOT reply with ALPS unless it is also negotiating ALPN. The ALPS payload is protocol-dependent, and as such it MUST be specified with respect to a selected ALPN.

For application protocols that support 0-RTT data, both the client and the server have to remember the settings provided by the both sides during the original connection. If the client sends 0-RTT data and the server accepts it, the ALPS values SHALL be the same values as were during the original connection. In all other cases (including session resumption that does not result in server accepting early data), new ALPS values SHALL be negotiated.

If the client wishes to send different client settings for the 0-RTT session, it MUST NOT offer 0-RTT. Conversely, if the server would send different server settings, it MUST reject 0-RTT. Note that the ALPN itself is similarly required to match the one in the original connection, thus the settings only need to be remembered or checked for a single application protocol.

4. Wire protocol

ALPS is only supported in TLS version 1.3 or later, as the earlier versions do not provide any confidentiality protections for the handshake data. The exchange is performed in three steps:

1. The client sends an extension in ClientHello that enumerates all ALPN values for which ALPS is supported.

2. The server sends an encrypted extension containing the server settings.

3. The client sends a new handshake message containing the client settings.

```
     Client                                               Server

     ClientHello
     + alpn
     + alps                    -------->

                                                     ServerHello
                                           {EncryptedExtensions}
                                                        + {alpn}
                                                        + {alps}
                                                             ...
                               <--------              {Finished}

     {ClientApplicationSettings}
     {Certificate*}
     {CertificateVerify*}
     {Finished}                -------->

                  +  Indicates extensions sent in the
                     previously noted message.

                  {} Indicates messages protected using
                     the handshake keys.

                  *  Indicates optional messages that are
                     not related to ALPS.
```

Figure 1: ALPS exchange in a full TLS handshake

A TLS client can enable ALPS by specifying an "application_settings" extension. The value of the "extension_data" field for the ALPS extension SHALL be a ApplicationSettingsSupport struct:

```
    struct {
        ProtocolName supported_protocols<2..2^16-1>;
    } ApplicationSettingsSupport;
```

Here, the "supported_protocols" field indicates the names of the protocols (as defined in [RFC7301]) for which ALPS exchange is supported; this is necessary for the situations when the client offers multiple ALPN values but only supports ALPS in some of them.

If the server chooses an ALPN value for which the client has offered ALPS support, the server MAY send an "application_settings" extension in the EncryptedExtensions. The value of the "extension_data" field in that case SHALL be an opaque blob containing the server settings as specified by the application protocol.

If the client receives an EncryptedExtensions message containing an "application_settings" extension from the server, after receiving server's Finished message it MUST send a ClientApplicationSettings handshake message before sending the Finished message:

```
    enum {
        client_application_settings(TBD), (255)
    } HandshakeType;

    struct {
        opaque application_settings<0..2^16-1>;
    } ClientApplicationSettings;
```

The value of the "application_settings" field SHALL be an opaque blob containing the client settings as specified by the application protocol. If the client is providing a client certificate, the ClientApplicationSettings message MUST precede the Certificate message sent by the client.

If the ClientApplicationSettings message is sent or received during the handshake, it SHALL be appended to the end of client's Handshake Context context as defined in Section 4.4 of [RFC8446]. In addition, for Post-Handshake Handshake Context, it SHALL be appended after the client Finished message.

When performing session resumption with 0-RTT data, the settings are carried over from the original connection. The server SHALL send an empty "application_settings" extension if it accepts 0-RTT, and the client SHALL NOT send a ClientApplicationSettings message.

5. Security Considerations

ALPS is protected using the handshake keys, which are the secret keys derived as a result of (EC)DHE between the client and the server.

In order to ensure that the ALPS values are authenticated, the TLS implementation MUST NOT reveal the contents of peer's ALPS until peer's Finished message is received, with exception of cases where the ALPS has been carried over from the previous connection.

6. IANA Considerations

IANA will update the "TLS ExtensionType Values" registry to include "application_settings" with the value of TBD; the list of messages in which this extension may appear is "CH, SH".

IANA will also update the "TLS HandshakeType" registry to include "client_application_settings" message with value TBD, and "DTLS-OK" set to "Y".

7. References

7.1. Normative References

[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>.

[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/info/rfc7301>.

[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>.

[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/info/rfc8446>.

7.2. Informative References

[I-D.ietf-quic-transport] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-29, 9 June 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-29.txt>.

[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>.

Acknowledgments

This document has benefited from contributions and suggestions from David Benjamin, Nick Harper, David Schinazi, Renjie Tang and many others.

Author's Address

Victor Vasiliev
Google

Email: vasilvv@google.com

TLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 25 March 2021

D. Benjamin
V. Vasiliev
Google

21 September 2020

TLS Application-Layer Protocol Settings Extension
draft-vvv-tls-alps-01

Abstract

This document describes a Transport Layer Security (TLS) extension for negotiating application-layer protocol settings (ALPS) within the TLS handshake. Any application-layer protocol operating over TLS can use this mechanism to indicate its settings to the peer in parallel with the TLS handshake completion.

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/ (https://mailarchive.ietf.org/arch/browse/tls/).

Source for this draft and an issue tracker can be found at https://github.com/vasilvv/tls-alps (https://github.com/vasilvv/tls-alps).

Status of This Memo

This Internet-Draft will expire on 25 March 2021.

Copyright Notice

Table of Contents

1. Introduction

An application-layer protocol often starts with both parties negotiating parameters under which the protocol operates; for instance, HTTP/2 [RFC7540] uses a SETTINGS frame to exchange the list of protocol parameters supported by each endpoint. This is usually achieved by waiting for TLS handshake [RFC8446] to complete and then performing the application-layer handshake within the application protocol itself. This approach, despite its apparent simplicity at first, has multiple drawbacks:

1. While the server is technically capable of sending configuration to the peer as soon as it sends its Finished message, most TLS implementations do not allow any application data to be sent until the Finished message is received from the client. This adds an extra round-trip to the time of when the server settings are available to the client.

2. In QUIC, any settings delivered within the application layer can arrive after other application data; thus, the application has to operate under the assumption that peer’s settings are not always available.

3. If the application needs to be aware of the server settings in order to send 0-RTT data, the application has to manually integrate with the TLS stack to associate the settings with TLS session tickets.

This document introduces a new TLS extension, "application_settings", that allows applications to exchange settings within the TLS handshake. Through doing that, the settings can be made available to the application as soon as the handshake completes, and can be associated with TLS session tickets automatically at the TLS layer. This approach allows the application protocol to be designed with the assumption that it has access to the peer’s settings whenever it is able to send data.

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. Semantics

Settings are defined to be an opaque blob that is specified by the application when initiating a TLS connection. The settings are meant to be a _declaration_ of the protocol parameters supported by the sender. While in this version of the extension the server settings are always sent first, this may change in future versions; thus, the application MUST NOT vary client settings based on the ones received from the server.

ALPS is _not_ a negotiation mechanism: there is no notion of rejecting peer’s settings, and the settings are not responses to one another. Nevertheless, it is possible for parties to coordinate behavior by, for instance, requiring a certain parameter to be present in both client and server settings. This makes ALPS mechanism similar to QUIC transport parameters [I-D.ietf-quic-transport] or HTTP/2 SETTINGS frame [RFC7540], but puts it in contrast to similar mechanisms in TLS.

Settings are exchanged as a part of the TLS handshake that is encrypted with the handshake keys. When the server settings are sent, the identity of the client has not been yet established; therefore, an application MUST NOT use ALPS if it requires the settings to be available only to the authenticated clients.

The ALPS model provides applications with a guarantee that the settings are available before any application data can be written. Note that this implies that when the full handshake is performed, the server can no longer send data immediately after sending its Finished message; it has to wait for the client to respond with its settings. This may negatively impact the latency of the protocols where the server sends the first message, however it should be noted that sending application data before receiving has not been widely supported by TLS implementations, nor has it been allowed in situations when establishing client identity through TLS is required.

ALPS can only be used in conjunction with Application-Layer Protocol Negotiation: the client MUST offer ALPN [RFC7301] if advertising ALPS support, and the server MUST NOT reply with ALPS unless it is also negotiating ALPN. The ALPS payload is protocol-dependent, and as such it MUST be specified with respect to a selected ALPN.

4. Wire Protocol

ALPS is only supported in TLS version 1.3 or later, as the earlier versions do not provide any confidentiality protections for the handshake data. The exchange is performed in three steps:

1. The client sends an extension in ClientHello that enumerates all ALPN values for which ALPS is supported.

2. The server sends an encrypted extension containing the server settings.

3. The client sends an encrypted extension containing the client settings.

```
       Client                                                 Server

       ClientHello
       + alpn
       + alps                    -------->

                                                         ServerHello
                                               {EncryptedExtensions}
                                                              + alpn
                                                              + alps
                                                                 ...
                                 <--------                {Finished}

       {EncryptedExtensions}
       + alps
       {Certificate*}
       {CertificateVerify*}
       {Finished}                -------->

                   +  Indicates extensions sent in the
                      previously noted message.

                   {} Indicates messages protected using
                      the handshake keys.

                   *  Indicates optional messages that are
                      not related to ALPS.
```

Figure 1: ALPS exchange in a full TLS handshake

A TLS client can enable ALPS by specifying an "application_settings" extension in the ClientHello message. The value of the "extension_data" field for this extension SHALL be a ApplicationSettingsSupport struct:

```
    struct {
        ProtocolName supported_protocols<2..2^16-1>;
    } ApplicationSettingsSupport;
```

Here, the "supported_protocols" field indicates the names of the protocols (as defined in [RFC7301]) for which ALPS exchange is supported; this is necessary for the situations when the client offers multiple ALPN values but only supports ALPS in some of them.

If the server chooses an ALPN value for which the client has offered ALPS support, the server MAY negotiate ALPS by sending an "application_settings" extension in its EncryptedExtensions message. The value of the "extension_data" field in that case SHALL be an opaque blob containing the server settings as specified by the application protocol.

If the client receives an EncryptedExtensions message containing an "application_settings" extension from the server, it MUST send an EncryptedExtensions message (see Section 4.1) containing an "application_extensions" extension. The value of the "extension_data" in this extension SHALL be an opaque blob containing the client settings as specified by the application protocol. A server which negotiates ALPS MUST abort the handshake with a "missing_extension" alert if the client's EncryptedExtensions is missing this extension.

4.1. Client Encrypted Extensions

This specification introduces the client EncryptedExtensions message. The format and HandshakeType code point match the server EncryptedExtensions message. When sent, it is encrypted with handshake traffic keys and sent by the client after receiving the server Finished message and before the client sends the Certificate, CertificateVerify (if any), and Finished messages. It SHALL be appended to the Client Handshake Context, as defined Section 4.4 of [RFC8446]. It additionally SHALL be inserted after the server Finished in the Post-Handshake Handshake Context.

The client MUST send the EncryptedExtensions message if any extension sent in the server EncryptedExtension message contains the CEE token in the TLS 1.3 column of the TLS ExtensionType Values registry. Otherwise, the client MUST NOT send the message. The server MUST abort the handshake with a "unexpected_message" alert if the message was sent or omitted incorrectly.

The client MAY send an extension in the client EncryptedExtension message if that extension's entry in the registry contains a CEE token and the server EncryptedExtensions message included the extension. Otherwise, the client MUST NOT send the extension. If a server receives an extension which does not meet this criteria, it MUST abort the handshake with an "unsupported_extension" alert.

Future extensions MAY use the client EncryptedExtensions message by including the CEE token in the TLS 1.3 registry. The above rules ensure clients will not send EncryptedExtensions messages to older servers, but will send EncryptedExtensions when some negotiated extension uses it.

[[TODO: Section 4.6.1 of RFC8446 allows the server to predict the client Finished flight and send a ticket early. This is still possible with 0-RTT handshakes here because we omit rather than repeat the redudant ALPS information, but, in the general extension case, client EncryptedExtensions breaks this. Extension order is unpredictable. We should resolve this conflict, either by dropping that feature or removing flexibility here.]]

4.2. 0-RTT Handshakes

ALPS ensures settings are available before reading and writing application data, so handshakes which negotiate early data instead use application settings from the PSK. To use early data with a PSK, the TLS implementation MUST associate both client and server application settings, if any, with the PSK. For a resumption PSK, these values are determined from the original connection. For an external PSK, this values should be configured with it. Existing PSKs are considered to not have application settings.

If the server accepts early data, the server SHALL NOT send an "application_settings" extension, and thus the client SHALL NOT send a "application_settings" extension in its EncryptedExtensions message. Unless the server has sent some other extension which uses client EncryptedExtensions, the client SHALL NOT send an EncryptedExtensions message. Instead, the connection implicitly uses the PSK's application settings, if any.

If the server rejects early data, application settings are negotiated independently of the PSK, as if early data were not offered.

If the client wishes to send different client settings for the connection, it MUST NOT offer 0-RTT. Conversely, if the server wishes to use send different server settings, it MUST reject 0-RTT. Note that the ALPN itself is similarly required to match the one in the original connection, thus the settings only need to be remembered or checked for a single application protocol. Implementations are RECOMMENDED to first determine the desired application protocol and settings independent of early data, and then decline to offer or accept early data if the values do not match the PSK. This preserves any ALPN and ALPS configuration specified by the calling application.

5. Security Considerations

ALPS is protected using the handshake keys, which are the secret keys derived as a result of (EC)DHE between the client and the server.

In order to ensure that the ALPS values are authenticated, the TLS implementation MUST NOT reveal the contents of peer's ALPS until peer's Finished message is received, with exception of cases where the ALPS has been carried over from the previous connection.

6. IANA Considerations

IANA will update the "TLS ExtensionType Values" registry to include "application_settings" with the value of TBD; the list of messages in which this extension may appear is "CH, EE, CEE".

7. References

7.1. Normative References

[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>.

[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/info/rfc7301>.

[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>.

[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/info/rfc8446>.

7.2. Informative References

[I-D.ietf-quic-transport] Iyengar, J. and M. Thomson, "QUIC: A UDP-Based Multiplexed and Secure Transport", Work in Progress, Internet-Draft, draft-ietf-quic-transport-30, 9 September 2020, <http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-30.txt>.

[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>.

Acknowledgments

This document has benefited from contributions and suggestions from Nick Harper, David Schinazi, Renjie Tang and many others.

Authors' Addresses

David Benjamin
Google

Email: davidben@google.com

Victor Vasiliev
Google

Email: vasilvv@google.com

TLS Working Group
Internet-Draft
Intended status: Standards Track
Expires: 13 January 2021

V. Vasiliev
Google

12 July 2020

Transport Layer Security (TLS) Resumption across Server Names
draft-vvv-tls-cross-sni-resumption-00

Abstract

This document specifies a way for the parties in the Transport Layer Security (TLS) protocol to indicate that an individual session ticket can be used to perform resumption even if the Server Name of the new connection does not match the Server Name of the original.

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/ (https://mailarchive.ietf.org/arch/browse/tls/).

Source for this draft and an issue tracker can be found at https://github.com/vasilvv/tls-cross-sni-resumption (https://github.com/vasilvv/tls-cross-sni-resumption).

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Transport Layer Security protocol [RFC8446] allows the clients to use an abbreviated handshake in cases where the client has previously established a secure session with the same server. This mechanism is known as "session resumption", and its positive impact on performance makes it desirable to be able to use it as frequently as possible.

Modern application-level protocols, HTTP in particular, often require accessing multiple servers within a single workflow. Since the identity of the server is established through its certificate, in the ideal case, the resumption would be possible to all of the domains for which the certificate is valid (see [PERF] for a survey of potential practical impact of such approach). TLS, starting with version 1.3, defines the SNI value to be a property of an individual connection that is not retained across sessions ([RFC8446], Section 4.2.11). However, in the absence of additional signals, it discourages using a session ticket when the SNI value does not match ([RFC8446], Section 4.6.1), as there is normally no reason to assume that all servers sharing the same certificate would also share the same session keys. The extension defined in this document allows the server to provide such a signal in-band.

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. The Extension

The server MAY send a resumption_across_names(TBD) extension in a NewSessionTicket message. That extension SHALL have an empty body. If the extension is sent, it indicates that the client MAY use the ticket for any SNI value for which the certificate presented by the server is valid. The server MUST handle the ticket correctly by either resuming and using a new SNI provided by the client, or by ignoring the ticket.

The server MAY send the extension if it reasonably believes that any server for any identity presented in its certificate would be capable of accepting that ticket. The server SHOULD NOT send the extension otherwise, since, if the client follows the single-use ticket policy recommended by [RFC8446], sending the ticket results in it being no longer usable regardless of whether resumption has succeeded.

4. Security Considerations

This document does not alter any of the security requirements of [RFC8446], but merely lifts a performance-motivated "SHOULD NOT" recommendation from Section 4.6.1. Notably, it still relies on the server certificate being re-validated against the new SNI at the session resumption time.

If a client certificate has been associated with the session, the client MUST use the same policy on whether to present said certificate to the server as if it were a new TLS session. For instance, if the client would show a certificate choice prompt for every individual domain it connects to, it MUST show that prompt for the new host when performing cross-domain resumption.

Cross-domain resumption, like other similar mechanisms (e.g. cross-domain HTTP connection reuse), can incentivize the server deployments to create server certificates valid for a wider range of domains than they would otherwise. However, any increase in the scope of a certificate comes at a cost: the wider is the scope of the certificate, the wider is the impact of the key compromise for that certificate. In addition, creating a certificate that is valid for multiple hostnames can lead to complications if some of those hostnames change ownership, or otherwise require a different operational domain.

5. IANA Considerations

IANA (will add/has added) the following entry to the "TLS ExtensionType Values" table of the "Transport Layer Security (TLS) Extensions" registry:

Value TBD

Extension Name resumption_across_names

TLS 1.3 NST

Recommended N

Reference This document

6. References

6.1. Normative References

[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>.

[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/info/rfc8446>.

6.2. Informative References

[PERF] Sy, E., Moennich, M., Mueller, T., Federrath, H., and M. Fischer, "Enhanced Performance for the encrypted Web through TLS Resumption across Hostnames", 7 February 2019, <https://arxiv.org/pdf/1902.02531.pdf>.

Acknowledgments

Cross-name resumption has been previously implemented in the QUIC Crypto protocol as a preloaded list of hostnames.

Erik Sy has previously proposed a similar mechanism for TLS, draft-sy-tls-resumption-group (https://datatracker.ietf.org/doc/draft-sy-tls-resumption-group/). This document incorporates ideas from that draft.

This document has benefited from contributions and suggestions from David Benjamin, Nick Harper, David Schinazi, Ryan Sleevi, Ian Swett and many others.

Author's Address

Victor Vasiliev
Google

Email: vasilvv@google.com