Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 10 September 2020

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

9 March 2020

Delegated Credentials for TLS
draft-ietf-tls-subcerts-07

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.

Status of This Memo

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 authentication schemes for which the CA will issue credentials.

These dependencies cause problems in practice. Server operators often want to create short-lived certificates for servers in low-trust zones such as Content Delivery Networks (CDNs) or remote data centers. This allows server operators to limit the exposure of keys in cases where they do not realize a compromise has occurred. However, the risk inherent in cross-organizational transactions makes it operationally infeasible to rely on an external CA for such short-lived credentials. For instance, in the case of Online Certificate Status Protocol (OCSP) stapling (i.e., using the Certificate Status extension type ocsp [RFC8446]), a CA may fail to deliver OCSP stapled response. While this will result in degraded performance, the ramifications of failing to deliver short-lived certificates are even worse: the service that depends on those certificates would go down entirely. Thus, ensuring independence from CAs for short-lived certificates is critical to the uptime of a service.

To remove these dependencies, 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. Because the above problems do not relate to the CA's inherent function of validating possession of names, it is safe to make such delegations as long as they 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-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 secret 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 normal 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 intitiator 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's working key for the TLS hadshake

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.

It is possible to address the short-lived certificate concerns above by automating certificate issuance, e.g., with Automated Certificate Managment Encvironment (ACME) [RFC8555]. In addition to requiring frequent operationally-critical interactions with an external party, this makes the server operator dependent on the CA's willingness to issue certificates with sufficiently short lifetimes. It also fails to address the issues with algorithm support. Nonetheless, existing automated issuance APIs like ACME may be useful for provisioning credentials within an operator network.

4. Delegated Credentials

While X.509 forbids end-entity certificates from being used as issuers for other certificates, it is perfectly fine 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: Relative time in seconds from the beginning of the delegation certificate's notBefore value after which the delegated credential is no longer valid.

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. Minimizing their semantics in this way is intended to mitigate the risk of cross protocol attacks involving delegated credentials.

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 in the usual way. It also takes the following steps:

1. Verify that the current time is within the validity interval of the credential and that the credential's time to live 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.

2. 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.

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

4. 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]).

5. 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 Identfiier (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.

6. Security Considerations

6.1. Security of delegated private key

Delegated credentials limit the exposure of the TLS private key by limiting its validity. An attacker who compromises the private key of a delegated credential can act as a man-in-the-middle until the delegate 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 private key is thus important and several access control mechanisms SHOULD be used to protect it, including file system controls, physical security, or hardware security modules.

6.2. Re-use of delegated credentials in multiple contexts

It is possible to use the same delegated credential for both client and server authentication if the Certificate allows it. This is safe because the context string used for delegated credentials is distinct in both contexts.

6.3. 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.

6.4. Interactions with session resumption

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

6.5. 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. Acknowledgements

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

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>.

[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.

8.2. Informative References

[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>.

[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>.

[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