Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 18, 2019

R. Barnes
Mozilla
S. Iyengar
Facebook
N. Sullivan
Cloudflare
E. Rescorla
RTFM, Inc.

August 17, 2018

Delegated Credentials for TLS
draft-ietf-tls-subcerts-02

Abstract

The organizational separation between the operator of a TLS server 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 clients that do not support this specification.

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 February 18, 2019.

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 CDNs or remote data centers. This allows server

operators to limit the exposure of keys in cases that they do not realize a compromise has occurred. The risk inherent in cross-organizational transactions makes it operationally infeasible to rely on an external CA for such short-lived credentials. In OCSP stapling (i.e., using the Certificate Status extension types ocsp [RFC6066] or ocsp_multi [RFC6961]), if an operator chooses to talk frequently to the CA to obtain stapled responses, then failure to fetch an OCSP stapled response results only in degraded performance. On the other hand, failure to fetch a potentially large number of short lived certificates would result in the service not being available, which creates greater operational risk.

To remove these dependencies, this document proposes a limited delegation mechanism that allows a TLS server operator 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".

1.1. Change Log

(*) indicates changes to the wire protocol.

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.

2. 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 TLS server operator. The secret key used to sign a credential corresponds to the public key of the TLS server's X.509 end-entity certificate.

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

- The client provides an extension in its ClientHello that indicates support for this mechanism.
- The server provides both the certificate chain terminating in its certificate as well as the delegated credential.
- The client uses information in the server's certificate to verify the delegated credential and that the server is asserting an expected identity.
- The client uses the public key in the credential as the server's working key for the TLS handshake.

As detailed in Section 3, the delegated credential is cryptographically bound to the end-entity certificate and the protocol in 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 explicitly disallowed.

Delegated credentials allow the server 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 a delegated credential is compromised, servers may not issue credentials with a validity period longer than 7 days. This mechanism is described in detail in Section 3.1.

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 3.2.)

2.1. 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 which should not conflict with X.509 semantics.

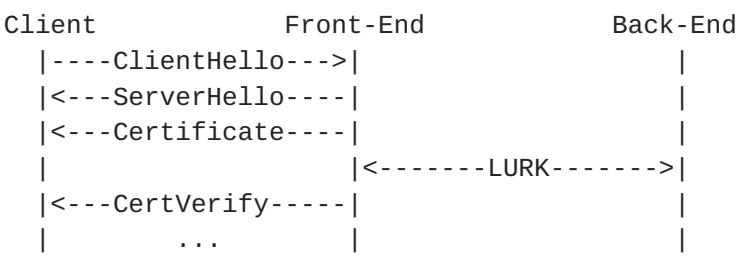
- Proxy certificates rely on the certificate path building process to establish a binding between the proxy certificate and the server certificate. Since the certificate path building process is not cryptographically protected, it is possible that a proxy certificate could be bound to another certificate 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 version of TLS and signature algorithm. This prevents them from being used for other protocols or with other signature algorithms than service owner allows.

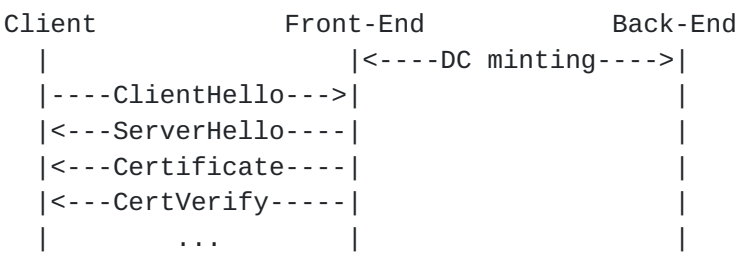
2.2. Related Work

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., LURK [I-D.mglt-lurk-tls-requirements]). 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 [I-D.ietf-tls-tls13], where DC is delegated credentials.

LURK:



Delegated credentials:



These two mechanisms can be complementary. A server could use credentials for clients that support them, while using LURK to support legacy clients.

It is possible to address the short-lived certificate concerns above by automating certificate issuance, e.g., with ACME [I-D.ietf-acme-acme]. 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.

3. 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;
     ProtocolVersion expected_version;
     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.

expected_version: The version of TLS in which the credential will be used, where the type ProtocolVersion is as defined in [RFC8446]. This is expected to match the protocol version that is negotiated by the client and server.

ASN1_subjectPublicKeyInfo: The credential's public key, a DER-encoded [X690] 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 signature over the credential with the end-entity certificate's public key, using the scheme.

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".

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.algorithm.

6. DelegatedCredential.scheme.

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, protocol, 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 to create and verify delegated credentials would be localized to the TLS stack, which has the advantage of avoiding changes to security-critical and often delicate PKI code (though of course moves that complexity to the TLS stack).

3.1. Client and Server behavior

This document defines the following extension code point.

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

A client which supports this specification SHALL send an empty "delegated_credential" extension in its ClientHello. 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. A delegated credential MUST NOT be provided unless a Certificate message is also sent. 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 algorithm and expected_cert_verify_algorithm fields MUST be of a type advertised by the client in the "signature_algorithms"

extension. A delegated credential MUST NOT be negotiated otherwise, even if the client advertises support for delegated credentials.

On receiving a delegated credential and a certificate chain, the client validates the certificate chain and matches the end-entity certificate to the server's expected identity following its normal procedures. 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 7 days.

2. Verify that expected_cert_verify_algorithm matches the scheme indicated in the server's CertificateVerify message.

3. Verify that expected_version matches the protocol version indicated in the server's "supported_versions" extension.

4. Verify that the end-entity certificate satisfies the conditions specified in Section 3.2.

5. Use the public key in the server'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 that receive invalid delegated credentials MUST terminate the connection with an "illegal_parameter" alert. If successful, the client uses the public key in the credential to verify the signature in the server's CertificateVerify message.

3.2. 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.

```
id-ce-delegationUsage OBJECT IDENTIFIER ::= { 1.3.6.1.4.1.44363.44 }
DelegationUsage ::= NULL
```

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]).

4. IANA Considerations

TBD

5. Security Considerations

5.1. 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 such as file system controls, physical security or hardware security modules.

5.2. 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.

5.3. 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.

6. Acknowledgements

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

7. References

7.1. Normative References

[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>.

[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>.

[X690] 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 8825-1:2002, 2002.

7.2. Informative References

[I-D.ietf-acme-acme]
Barnes, R., Hoffman-Andrews, J., McCarney, D., and J. Kasten, "Automatic Certificate Management Environment (ACME)", draft-ietf-acme-acme-14 (work in progress), August 2018.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-28 (work in progress), March 2018.

[I-D.mglt-lurk-tls-requirements]
Migault, D. and K. Ma, "Authentication Model and Security Requirements for the TLS/DTLS Content Provider Edge Server Split Use Case", draft-mglt-lurk-tls-requirements-00 (work in progress), January 2016.

[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>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <https://www.rfc-editor.org/info/rfc6066>.

[RFC6961] Pettersen, Y., "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension", RFC 6961, DOI 10.17487/RFC6961, June 2013, <https://www.rfc-editor.org/info/rfc6961>.

[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.

Authors' Addresses

Richard Barnes
Mozilla

Email: rlb@ipv.sx

Subodh Iyengar
Facebook

Email: subodh@fb.com

Nick Sullivan
Cloudflare

Email: nick@cloudflare.com

Eric Rescorla
RTFM, Inc.

Email: ekr@rtfm.com