Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 8, 2017

E. Rescorla
RTFM, Inc.
R. Barnes
Mozilla
S. Iyengar
Facebook
N. Sullivan
CloudFlare Inc.
July 7, 2016

Delegated Credentials for TLS
draft-rescorla-tls-subcerts-00

Abstract

The organizational separation between the operator of a TLS server and the certificate authority that provides it credentials can cause problems, for example when it comes to reducing the lifetime of certificates or supporting new cryptographic algorithms. This document describes a mechanism to allow TLS server operators to create their own credential delegations 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 http://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 January 8, 2017.

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) [RFC5246] [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. The risk inherent in cross-organizational transactions makes it infeasible to rely on an external CA for such short-lived credentials.

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 CAs 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" and the one issued by the operator as a "delegated credential".

[[Ed. - We use the phrase "credential" for the sub-certificates since it's an open issue whether they will be certificates or not.]]

[[Ed. - This document is framed as a single solution, because it would be best for the WG to ultimately settle on one solution to this problem. However, to facilitate discussion, we outline several possible options for how credential can be realized. Once the WG agrees on an overall approach, this draft will be revised to provide more details of that approach.]]

2. Solution Overview

A delegated credential is a digitally signed data structure with the following semantic fields:

- A validity interval
- A public key (with its associated algorithm)
- A list of valid server names

The signature on the credential indicates a delegation from the certificate which is issued to the TLS server operator. The key pair used to sign a credential is presumed to be one whose public key is contained in an X.509 certificate that associates one or more names to the credential.

A TLS handshake that uses 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 credential.
- The client uses information in the server's certificate to verify the signature on the credential and verify 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.

- The client uses the list of valid server names in the credential to verify that the credential is valid for the server.

The credential signature is subject to the negotiated signature algorithms. A credential cannot be used if the client advertises support for credentials however a server does not have a certificate which is compatible with any of the negotiated signature algorithms.

It was noted by [J"{a}ger et al.] 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, clients MUST NOT accept connections from certificates with the keyEncipherment KeyUsage.

[[Nick - This is a much less stringent requirement than a new flag, since it means that all existing ECDSA certificates can be re-used.]]

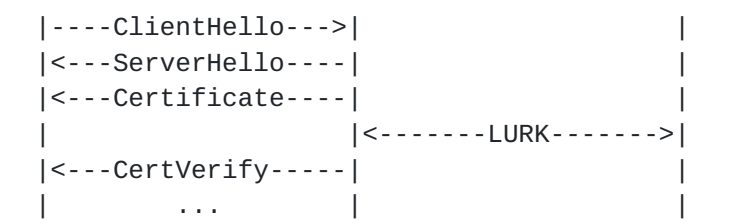
Credentials allow the server to terminate TLS connections on behalf of the certificate owner. If a credential is stolen, there is no mechanism for revoking it without revoking the certificate itself. To limit the exposure of a delegation credential compromise, servers MUST NOT issue credentials with a validity period longer than 7 days. Clients MUST NOT accept credentials with longer validity periods. [[TODO: which alert should the client send?]]

[[Ed. - The specifics of how credentials are structured and provided by the server are still to be determined; see below.]]

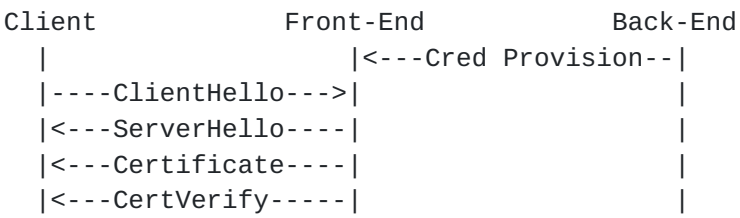
3. Related Work

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].

LURK:



Delegated credentials:



These two classes of mechanism 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.

4. Client Behavior

This document defines the following extension code point.

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

A client which supports this document SHALL send an empty "delegated_credential" extension in its ClientHello.

[[Option 1]] A server MUST NOT send this extension. If the extension is present, the server MAY send a credential. If the extension is not present, the server MUST NOT send a credential. A credential MUST NOT be provided unless a Certificate message is also sent. [[Option 2]] If the extension is present, the server MAY send a "delegated credential" extension containing the credential in the response. If the extension is not present, the server MUST NOT send a credential. A credential MUST NOT be provided unless a Certificate message is also sent.

On receiving a 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 then takes the following additional steps:

- Verify that the current time is within the validity interval of the credential
- Verify that the server name is included in the credential
- Use the public key in the server's end-entity certificate to verify the signature on the credential
- Use the public key in the credential to verify the CertificateVerify message provided in the handshake

[[Ed. - This will need to be updated if the sub-certificate can be restricted to a subset of the names in the master certificate.]]

5. Delegated Credentials

[[Ed. - This section is currently a sketch, intended to lay out the design space to facilitate discussion]]

Credentials obviously need to have some defined structure. It is possible to re-use X.509, but it may be better to define something new.

The format question also mostly decides the question of how the credential will be signed and delivered to the client. If the credential is an X.509 certificate, then it will be signed in that format, and probably provided in the TLS Certificate message as the end-entity certificate. If some new structure is devised, then it will need to define a signature method, and it will probably make more sense to carry it in a TLS extension.

The delivery mechanism is mostly a trivial question, but given that the server is switching between a normal certificate chain and one

including a credential based on a ClientHello extension, there could be some impact on the ease of implementation. For example, it may be easier to change the extensions in the ServerHello than to switch the certificate chain, or alternately it may be easier to simply let the server operator provide a whole chain terminating in the credential, depending on how much sanity checking the server does.

5.1. **Option 1a.** Name Constraints

It would be consistent with the requirements above to realize credentials as sub-certificates by having the CA issue a subordinate CA certificate to the TLS server operator, with a nameConstraints extension encoding the names the server operator is authorized for. Then the credentials would simply be normal end-entity certificates issued under this subordinate.

In order for this solution to be safe the subordinate CA certificate needs to have a critical nameConstraints extension. Historically, this solution has been unworkable due to legacy clients that could not process name constraints. However, since in this case we require the client to indicate support, it may be possible to have critical name constraints without compatibility impact.

Pro:

- Re-use existing issuance and validation code
- No change to client certificate validation and CertificateVerify processing

Con:

- Requires server operator to get a name-constrained subordinate CA certificate
- Name constraints are not universally recognized
- X.509 provides much richer semantics than required

5.2. **Option 1b.** End Entities as Issuers

One could also imagine a scheme in which the server could use an end-entity certificate as the issuer for a sub-certificate. Since servers are typically issued end-entity certificates by CAs, this could align better with CA issuance practices.

It's important to note that this would not enable existing end-entity certificates to be used to issue sub-certificates. That would create

risks such as those noted in [J"{a}ger et al.]. So there would be a need to define some marker that would be inserted into an end-entity certificate to indicate that it could be used to issue sub-certificates. It may be enough to require the certificate to not contain a keyEncipherment KeyUsage.

Pro:

- No change to client CertificateVerify processing (still uses last cert in the chain)

Con:

- Violates the semantics of the CA bit in basicConstraints
- Requires change to X.509 validation logic to allow sub-certificates
- X.509 provides much richer semantics than required

5.3. **Option 2.** Define a New Structure

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 key usage (RFC5280 section 4.2.1.3). We could define a new signed object format that would encode only the semantics that are needed for this application. For example, the TLS "digitally-signed" structure could be used:

```
digitally-signed struct {
  uint64 notBefore;
  uint64 notAfter;
  SignatureScheme algorithm;
  ServerName serverName;
  opaque publicKey<0..2^24-1>;
} DelegatedCredential;
```

The ServerNameList can be defined as follows (similar to RFC 6066 section 3): ~~~~~~~~~~ struct { NameType name_type; select (name_type) { case host_name: HostName; } name; } ServerName;

```
enum { host_name(0), (255) } NameType;

opaque HostName<1..2^16-1>; ~~~~~~~~~~
```

This would avoid any mis-match in semantics with X.509, and would likely require more processing code in the client. The code changes

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).

Pro:

- No change to client certificate validation
- No risk of conflict with X.509 semantics

Con:

- Requires new logic for generating and verifying credentials
- Requires changes to client CertificateVerify processing

5.4. Re-Use of the Master Certificate

[[OPEN ISSUE]]

The certificate can be configured so that it is usable directly as a TLS end-entity certificate (this is the natural design for Option 2) or alternately can be configured so that it is not acceptable as a TLS server certificate but rather can only be used for signing sub-certificates. In the former case, the server operator need only have one certificate, but with the risk that if the TLS server is compromised the attacker could issue themselves an arbitrary number of sub-certificates. Conversely, the master certificate may be configured so that it is not directly usable, thus requiring the name-holder to get two certificates, one for signing credentials and one for use in its TLS server. This adds additional complexity for the operator but allows the master certificate to be offline.

6. IANA Considerations

7. Security Considerations

8. References

8.1. Normative References

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

[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, <http://www.rfc-editor.org/info/rfc5280>.

8.2. Informative References

[I-D.ietf-acme-acme]
Barnes, R., Hoffman-Andrews, J., and J. Kasten, "Automatic Certificate Management Environment (ACME)", draft-ietf-acme-acme-02 (work in progress), March 2016.

[I-D.ietf-tls-tls13]
Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-13 (work in progress), May 2016.

[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.

Authors' Addresses

Eric Rescorla
RTFM, Inc.

Email: ekr@rtfm.com

Richard Barnes
Mozilla

Email: rlb@ipv.sx

Subodh Iyengar
Facebook

Email: subodh@fb.com

Nick Sullivan
CloudFlare Inc.

Email: nick@cloudflare.com