Mutual TLS Profile for OAuth 2.0
draft-ietf-oauth-mtls-05

Abstract

   This document describes Transport Layer Security (TLS) mutual
   authentication using X.509 certificates as a mechanism for OAuth
   client authentication to the authorization sever as well as for
   certificate bound sender constrained access tokens.

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 May 16, 2018.

Copyright Notice

Table of Contents

1. Introduction

   This document describes Transport Layer Security (TLS) mutual
   authentication using X.509 certificates as a mechanism for OAuth
   client authentication to the authorization sever as well as for
   sender constrained access to OAuth protected resources.

   The OAuth 2.0 Authorization Framework [RFC6749] defines a shared
   secret method of client authentication but also allows for the
   definition and use of additional client authentication mechanisms
   when interacting directly with the authorization server. This
   document describes an additional mechanism of client authentication
   utilizing mutual TLS [RFC5246] certificate-based authentication,
   which provides better security characteristics than shared secrets.
   While [RFC6749] documents client authentication for requests to the
   token endpoint, extensions to OAuth 2.0 (such as Introspection
   [RFC7662] and Revocation [RFC7009]) define endpoints that also
   utilize client authentication and the mutual TLS methods defined
   herein are applicable to those endpoints as well.

   Mutual TLS sender constrained access to protected resources ensures
   that only the party in possession of the private key corresponding to
   the certificate can utilize the access token to get access to the
   associated resources. Such a constraint is unlike the case of the
   basic bearer token described in [RFC6750], where any party in
   possession of the access token can use it to access the associated
   resources. Mutual TLS sender constrained access binds the access
   token to the client's certificate thus preventing the use of stolen
   access tokens or replay of access tokens by unauthorized parties.

   Mutual TLS sender constrained access tokens and mutual TLS client
   authentication are distinct mechanisms, which are complementary but
   don't necessarily need to be deployed together.

1.1. Requirements Notation and Conventions

   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 RFC
   2119 [RFC2119].

1.2. Terminology

   This specification uses the following phrases interchangeably:

      Transport Layer Security (TLS) Mutual Authentication

      Mutual TLS

These phrases all refer to the process whereby a client presents its
X.509 certificate and proves possession of the corresponding private
key to a server when negotiating a TLS session. In TLS 1.2 [RFC5246]
this requires the client to send Client Certificate and Certificate
Verify messages during the TLS handshake and for the server to verify
these messages.

2. Mutual TLS for OAuth Client Authentication

This section defines, as an extension of OAuth 2.0, Section 2.3
[RFC6749], two distinct methods of using mutual TLS X.509 client
certificates as client credentials. The requirement of mutual TLS
for client authentication is determined by the authorization server
based on policy or configuration for the given client (regardless of
whether the client was dynamically registered or statically
configured or otherwise established).

In order to utilize TLS for OAuth client authentication, the TLS
connection between the client and the authorization server MUST have
been established or reestablished with mutual X.509 certificate
authentication (i.e. the Client Certificate and Certificate Verify
messages are sent during the TLS Handshake [RFC5246]).

For all requests to the authorization server utilizing mutual TLS
client authentication, the client MUST include the "client_id"
parameter, described in OAuth 2.0, Section 2.2 [RFC6749]. The
presence of the "client_id" parameter enables the authorization
server to easily identify the client independently from the content
of the certificate. The authorization server can locate the client
configuration using the client identifier and check the certificate
presented in the TLS Handshake against the expected credentials for
that client. The authorization server MUST enforce some method of
binding a certificate to a client. Sections Section 2.1 and
Section 2.2 below define two ways of binding a certificate to a
client as two distinct client authentication methods.

2.1. PKI Mutual TLS OAuth Client Authentication Method

The PKI (public key infrastructure) method of mutual TLS OAuth client
authentication uses a subject distinguished name (DN) and validated
certificate chain to identify the client. The TLS handshake is
utilized to validate the client's possession of the private key
corresponding to the public key in the certificate and to validate
the corresponding certificate chain. The client is successfully
authenticated if the subject information in the certificate matches
the expected DN configured or registered for that particular client.
The PKI method facilitates the way X.509 certificates are
traditionally being used for authentication. It also allows the

client to rotate its X.509 certificates without the need to modify
its respective authentication data at the authorization server by
obtaining a new certificate with the same subject DN from a trusted
certificate authority (CA).

2.1.1. PKI Authentication Method Metadata Value

The "OAuth Token Endpoint Authentication Methods" registry
[IANA.OAuth.Parameters] contains values, each of which specify a
method of authenticating a client to the authorization server. The
values are used to indicate supported and utilized client
authentication methods in authorization server metadata, such as
OpenID Connect Discovery [OpenID.Discovery] and OAuth 2.0
Authorization Server Metadata [I-D.ietf-oauth-discovery], and in the
OAuth 2.0 Dynamic Client Registration Protocol [RFC7591]. For the
PKI method of mutual TLS client authentication, this specification
defines and registers the following authentication method metadata
value.

tls_client_auth
   Indicates that client authentication to the authorization server
   will occur with mutual TLS utilizing the PKI method of associating
   a certificate to a client.

2.1.2. Client Registration Metadata

The following metadata parameter is introduced for the OAuth 2.0
Dynamic Client Registration Protocol [RFC7591] in support of the PKI
method of binding a certificate to a client:

tls_client_auth_subject_dn
   An [RFC4514] string representation of the expected subject
   distinguished name of the certificate the OAuth client will use in
   mutual TLS authentication.

2.2. Self-Signed Certificate Mutual TLS OAuth Client Authentication
       Method

This method of mutual TLS OAuth client authentication is intended to
support client authentication using self-signed certificates. As
pre-requisite, the client registers an X.509 certificate or a trusted
source for its X.509 certificates (such as the "jwks_uri" as defined
in [RFC7591]) with the authorization server. During authentication,
TLS is utilized to validate the client's possession of the private
key corresponding to the public key presented within the certificate
in the respective TLS handshake. In contrast to the PKI method, the
certificate chain is not validated in this case. The client is
successfully authenticated, if the subject public key info of the

certificate matches the subject public key info of one of the
certificates configured or registered for that particular client.
The Self-Signed Certificate method allows to use mutual TLS to
authenticate clients without the need to maintain a PKI. When used
in conjunction with a "jwks_uri" for the client, it also allows the
client to rotate its X.509 certificates without the need to change
its respective authentication data directly with the authorization
server.

2.2.1. Self-Signed Certificate Authentication Method Metadata Value

The "OAuth Token Endpoint Authentication Methods" registry
[IANA.OAuth.Parameters] contains values, each of which specify a
method of authenticating a client to the authorization server. The
values are used to indicate supported and utilized client
authentication methods in authorization server metadata, such as
OpenID Connect Discovery [OpenID.Discovery] and OAuth 2.0
Authorization Server Metadata [I-D.ietf-oauth-discovery], and in the
OAuth 2.0 Dynamic Client Registration Protocol [RFC7591]. For the
Self-Signed Certificate method of binding a certificate to a client
using mutual TLS client authentication, this specification defines
and registers the following authentication method metadata value.

self_signed_tls_client_auth
   Indicates that client authentication to the authorization server
   will occur using mutual TLS with the client utilizing a self-
   signed certificate.

2.2.2. Client Registration Metadata

For the Self-Signed Certificate method of binding a certificate to a
client using mutual TLS client authentication, the existing
"jwks_uri" or "jwks" metadata parameters from [RFC7591] are used to
convey client's certificates and public keys, where the X.509
certificates are represented using the JSON Web Key (JWK) [RFC7517]
"x5c" parameter (note that Sec 4.7 of RFC 7517 requires that the key
in the first certificate of the "x5c" parameter must match the public
key represented by other members of the JWK).

3. Mutual TLS Sender Constrained Resources Access

When mutual TLS is used at the token endpoint, the authorization
server is able to bind the issued access token to the client
certificate. Such a binding is accomplished by associating the
certificate with the token in a way that can be accessed by the
protected resource, such as embedding the certificate hash in the
issued access token directly, using the syntax described in
Section 3.1, or through token introspection as described in

Section 3.2. Other methods of associating a certificate with an
access token are possible, per agreement by the authorization server
and the protected resource, but are beyond the scope of this
specification.

The client makes protected resource requests as described in
[RFC6750], however, those requests MUST be made over a mutually
authenticated TLS connection using the same certificate that was used
for mutual TLS at the token endpoint.

The protected resource MUST obtain the client certificate used for
mutual TLS authentication and MUST verify that the certificate
matches the certificate associated with the access token. If they do
not match, the resource access attempt MUST be rejected with an error
per [RFC6750] using an HTTP 401 status code and the "invalid_token"
error code.

Metadata to convey server and client capabilities for mutual TLS
sender constrained access tokens is defined in Section 3.3 and
Section 3.4 respectively.

3.1. X.509 Certificate Thumbprint Confirmation Method for JWT

When access tokens are represented as JSON Web Tokens (JWT)[RFC7519],
the certificate hash information SHOULD be represented using the
"x5t#S256" confirmation method member defined herein.

To represent the hash of a certificate in a JWT, this specification
defines the new JWT Confirmation Method RFC 7800 [RFC7800] member
"x5t#S256" for the X.509 Certificate SHA-256 Thumbprint. The value
of the "x5t#S256" member is a base64url-encoded SHA-256[SHS] hash
(a.k.a. thumbprint or digest) of the DER encoding of the X.509
certificate[RFC5280] (note that certificate thumbprints are also
sometimes known as certificate fingerprints).

The following is an example of a JWT payload containing an "x5t#S256"
certificate thumbprint confirmation method.

```
   {
     "iss": "https://server.example.com",
     "sub": "ty.webb@example.com",
     "exp": 1493726400,
     "nbf": 1493722800,
     "cnf":{
       "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
     }
   }
```

Figure 1: Example claims of a Certificate Thumbprint Constrained JWT

If, in the future, certificate thumbprints need to be computed using
hash functions other than SHA-256, it is suggested that additional
related JWT confirmation methods members be defined for that purpose.
For example, a new "x5t#S512" (X.509 Certificate Thumbprint using
SHA-512) confirmation method member could be defined by registering
it in the the IANA "JWT Confirmation Methods" registry
[IANA.JWT.Claims] for JWT "cnf" member values established by
[RFC7800].

3.2. Confirmation Method for Token Introspection

OAuth 2.0 Token Introspection [RFC7662] defines a method for a
protected resource to query an authorization server about the active
state of an access token as well as to determine meta-information
about the token.

For a mutual TLS sender constrained access token, the hash of the
certificate to which the token is bound is conveyed to the protected
resource as meta-information in a token introspection response. The
hash is conveyed using the same structure as the certificate SHA-256
thumbprint confirmation method, described in Section 3.1, as a top-
level member of the introspection response JSON. The protected
resource compares that certificate hash to a hash of the client
certificate used for mutual TLS authentication and rejects the
request, if they do not match.

Proof-of-Possession Key Semantics for JSON Web Tokens [RFC7800]
defined the "cnf" (confirmation) claim, which enables confirmation
key information to be carried in a JWT. However, the same proof-of-
possession semantics are also useful for introspected access tokens
whereby the protected resource obtains the confirmation key data as
meta-information of a token introspection response and uses that
information in verifying proof-of-possession. Therefore this
specification defines and registers proof-of-possession semantics for
OAuth 2.0 Token Introspection [RFC7662] using the "cnf" structure.
When included as a top-level member of an OAuth token introspection

response, "cnf" has the same semantics and format as the claim of the same name defined in [RFC7800]. While this specification only explicitly uses the "x5t#S256" confirmation method member, it needed to define and register the higher level "cnf" structure as an introspection response member in order to define and use its more specific "x5t#S256" confirmation method.

The following is an example of an introspection response for an active token with an "x5t#S256" certificate thumbprint confirmation method.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "active": true,
  "iss": "https://server.example.com",
  "sub": "ty.webb@example.com",
  "exp": 1493726400,
  "nbf": 1493722800,
  "cnf":{
    "x5t#S256": "bwcK0esc3ACC3DB2Y5_lESsXE8o9ltc05O89jdN-dg2"
  }
}
```

Figure 2: Example Introspection Response for a Certificate
Constrained Access Token

3.3. Authorization Server Metadata

This document introduces the following new authorization server metadata parameter to signal the server's capability to issue certificate bound access tokens:

mutual_tls_sender_constrained_access_tokens
   OPTIONAL. Boolean value indicating server support for mutual TLS sender constrained access tokens. If omitted, the default value is "false".

3.4. Client Registration Metadata

The following new client metadata parameter is introduced to convey the client's intention to use certificate bound access tokens:

mutual_tls_sender_constrained_access_tokens

OPTIONAL. Boolean value used to indicate the client's intention
to use mutual TLS sender constrained access tokens. If omitted,
the default value is "false".

4. Implementation Considerations

4.1. Authorization Server

The authorization server needs to setup its TLS configuration
appropriately for the binding methods it supports.

If the authorization server wants to support mutual TLS client
authentication and other client authentication methods in parallel,
it should make mutual TLS optional.

If the authorization server supports the Self-Signed Certificate
method, it should configure the TLS stack in a way that it does not
verify whether the certificate presented by the client during the
handshake is signed by a trusted CA certificate.

The authorization server may also consider hosting the token
endpoint, and other endpoints requiring client authentication, on a
separate host name in order to prevent unintended impact on the TLS
behavior of its other endpoints, e.g. authorization or registration.

4.2. Resource Server

From the perspective of the resource server, TLS client
authentication is used as a proof of possession method only. For the
purpose of client authentication, the resource server may completely
rely on the authorization server. So there is no need to validate
the trust chain of the client's certificate in any of the methods
defined in this document. The resource server should therefore
configure the TLS stack in a way that it does not verify whether the
certificate presented by the client during the handshake is signed by
a trusted CA certificate.

4.3. Sender Constrained Access Tokens Without Client Authentication

This document allows use of client authentication only or client
authentication in combination with sender constraint access tokens.
Use of mutual TLS sender constrained access tokens without client
authentication (e.g. to support binding access tokens to a TLS client
certificate for public clients) is also possible. The authorization
server would configure the TLS stack in the same manner as for the
Self-Signed Certificate method such that it does not verify that the
certificate presented by the client during the handshake is signed by
a trusted CA. Individual instances of a public client would then

create a self-signed certificate for mutual TLS with the
authorization server and resource server. The authorization server
would not authenticate the client at the OAuth layer but would bind
issued access tokens to the certificate, which the client has proven
possession of the corresponding private key. The access token is
then mutual TLS sender constrained and can only be used by the client
possessing the certificate and private key and utilizing them to
negotiate mutual TLS on connections to the resource server.

4.4. Certificate Bound Access Tokens

As described in Section 3, an access token is bound to a specific
client certificate, which means that the same certificate must be
used for mutual TLS on protected resource access. It also implies
that access tokens are invalidated when a client updates the
certificate, which can be handled similar to expired access tokens
where the client requests a new access token (typically with a
refresh token) and retries the protected resource request.

5. IANA Considerations

5.1. JWT Confirmation Methods Registration

This specification requests registration of the following value in
the IANA "JWT Confirmation Methods" registry [IANA.JWT.Claims] for
JWT "cnf" member values established by [RFC7800].

o Confirmation Method Value: "x5t#S256"
o Confirmation Method Description: X.509 Certificate SHA-256
   Thumbprint
o Change Controller: IESG
o Specification Document(s): Section 3.1 of [[this specification]]

5.2. OAuth Authorization Server Metadata Registration

This specification requests registration of the following value in
the IANA "OAuth Authorization Server Metadata" registry
[IANA.OAuth.Parameters] established by [I-D.ietf-oauth-discovery].

o Metadata Name: "mutual_tls_sender_constrained_access_tokens"
o Metadata Description: Indicates authorization server support for
   mutual TLS sender constrained access tokens.
o Change Controller: IESG
o Specification Document(s): Section 3.3 of [[this specification]]

5.3. Token Endpoint Authentication Method Registration

This specification requests registration of the following value in
the IANA "OAuth Token Endpoint Authentication Methods" registry
[IANA.OAuth.Parameters] established by [RFC7591].

o Token Endpoint Authentication Method Name: "tls_client_auth"
o Change Controller: IESG
o Specification Document(s): Section 2.1.1 of [[this specification
   ]]

o Token Endpoint Authentication Method Name:
   "self_signed_tls_client_auth"
o Change Controller: IESG
o Specification Document(s): Section 2.2.1 of [[this specification
   ]]

5.4. OAuth Token Introspection Response Registration

This specification requests registration of the following value in
the IANA "OAuth Token Introspection Response" registry
[IANA.OAuth.Parameters] established by [RFC7662].

o Claim Name: "cnf"
o Claim Description: Confirmation
o Change Controller: IESG
o Specification Document(s): Section 3.2 of [[this specification]]

5.5. OAuth Dynamic Client Registration Metadata Registration

This specification requests registration of the following client
metadata definitions in the IANA "OAuth Dynamic Client Registration
Metadata" registry [IANA.OAuth.Parameters] established by [RFC7591]:

o Client Metadata Name:
   "mutual_tls_sender_constrained_access_tokens"
o Client Metadata Description: Indicates the client's intention to
   use mutual TLS sender constrained access tokens.
o Change Controller: IESG
o Specification Document(s): Section 3.4 of [[this specification]]

o Client Metadata Name: "tls_client_auth_subject_dn"
o Client Metadata Description: String value specifying the expected
   subject distinguished name of the client certificate.
o Change Controller: IESG
o Specification Document(s): Section 2.1.2 of [[this specification
   ]]

6. Security Considerations

6.1. TLS Versions and Best Practices

TLS 1.2 [RFC5246] is cited in this document because, at the time of
writing, it is the latest version that is widely deployed. However,
this document is applicable with other TLS versions supporting
certificate-based client authentication. Implementation security
considerations for TLS, including version recommendations, can be
found in Recommendations for Secure Use of Transport Layer Security
(TLS) and Datagram Transport Layer Security (DTLS) [BCP195].

6.2. X.509 Certificate Spoofing

If the PKI method is used, an attacker could try to impersonate a
client using a certificate for the same DN issued by another CA,
which the authorization server trusts. To cope with that threat, the
authorization server may decide to only accept a limited number of
CAs whose certificate issuance policy meets its security
requirements.

7. References

7.1. Normative References

[BCP195]   Sheffer, Y., Holz, R., and P. Saint-Andre,
            "Recommendations for Secure Use of Transport Layer
            Security (TLS) and Datagram Transport Layer Security
            (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May
            2015, <http://www.rfc-editor.org/info/bcp195>.

[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>.

[RFC4514]   Zeilenga, K., Ed., "Lightweight Directory Access Protocol
            (LDAP): String Representation of Distinguished Names",
            RFC 4514, DOI 10.17487/RFC4514, June 2006,
            <https://www.rfc-editor.org/info/rfc4514>.

[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>.

   [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>.

   [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework",
              RFC 6749, DOI 10.17487/RFC6749, October 2012,
              <https://www.rfc-editor.org/info/rfc6749>.

   [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization
              Framework: Bearer Token Usage", RFC 6750,
              DOI 10.17487/RFC6750, October 2012,
              <https://www.rfc-editor.org/info/rfc6750>.

   [RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-
              Possession Key Semantics for JSON Web Tokens (JWTs)",
              RFC 7800, DOI 10.17487/RFC7800, April 2016,
              <https://www.rfc-editor.org/info/rfc7800>.

   [SHS]   National Institute of Standards and Technology, "Secure
              Hash Standard (SHS)", FIPS PUB 180-4, March 2012,
              <http://csrc.nist.gov/publications/fips/fips180-4/
              fips-180-4.pdf>.

7.2. Informative References

   [I-D.ietf-oauth-discovery]
              Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0
              Authorization Server Metadata", draft-ietf-oauth-
              discovery-07 (work in progress), September 2017.

   [IANA.JWT.Claims]
              IANA, "JSON Web Token Claims",
              <http://www.iana.org/assignments/jwt>.

   [IANA.OAuth.Parameters]
              IANA, "OAuth Parameters",
              <http://www.iana.org/assignments/oauth-parameters>.

   [OpenID.Discovery]
              Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID
              Connect Discovery 1.0", August 2015,
              <http://openid.net/specs/
              openid-connect-discovery-1_0.html>.

   [RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth
              2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009,
              August 2013, <https://www.rfc-editor.org/info/rfc7009>.

   [RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517,
              DOI 10.17487/RFC7517, May 2015,
              <https://www.rfc-editor.org/info/rfc7517>.

   [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token
              (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015,
              <https://www.rfc-editor.org/info/rfc7519>.

   [RFC7591] Richer, J., Ed., Jones, M., Bradley, J., Machulak, M., and
              P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol",
              RFC 7591, DOI 10.17487/RFC7591, July 2015,
              <https://www.rfc-editor.org/info/rfc7591>.

   [RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection",
              RFC 7662, DOI 10.17487/RFC7662, October 2015,
              <https://www.rfc-editor.org/info/rfc7662>.

Appendix A. Acknowledgements

   Scott "not Tomlinson" Tomilson and Matt Peterson were involved in
   design and development work on a mutual TLS OAuth client
   authentication implementation that informed some of the content of
   this document.

   Additionally, the authors would like to thank the following people
   for their input and contributions to the specification: Sergey
   Beryozkin, Vladimir Dzhuvinov, Samuel Erdtman, Phil Hunt, Takahiko
   Kawasaki Sean Leonard, Kepeng Li, James Manger, Jim Manico, Nov
   Matake, Sascha Preibisch, Justin Richer, Dave Tonge, and Hannes
   Tschofenig.

Appendix B. Document(s) History

   [[to be removed by the RFC Editor before publication as an RFC]]

   draft-ietf-oauth-mtls-05

   o Editorial fixes

   draft-ietf-oauth-mtls-04

   o Change the name of the 'Public Key method' to the more accurate
      'Self-Signed Certificate method' and also change the associated

      authentication method metadata value to
      "self_signed_tls_client_auth".
   o Removed the "tls_client_auth_root_dn" client metadata field as
      discussed in https://mailarchive.ietf.org/arch/msg/oauth/
      swDV2y0be6o8czGKQi1eJV-g8qc
   o Update draft-ietf-oauth-discovery reference to -07
   o Clarify that MTLS client authentication isn't exclusive to the
      token endpoint and can be used with other endpoints, e.g. RFC
      7009 revocation and 7662 introspection, that utilize client
      authentication as discussed in
      https://mailarchive.ietf.org/arch/msg/oauth/
      bZ6mft0G7D3ccebhOxnEYUv4puI
   o Reorganize the document somewhat in an attempt to more clearly
      make a distinction between mTLS client authentication and
      certificate bound access tokens as well as a more clear
      delineation between the two (PKI/Public key) methods for client
      authentication
   o Editorial fixes and clarifications

   draft-ietf-oauth-mtls-03

   o Introduced metadata and client registration parameter to publish
      and request support for mutual TLS sender constrained access
      tokens
   o Added description of two methods of binding the cert and client,
      PKI and Public Key.
   o Indicated that the "tls_client_auth" authentication method is for
      the PKI method and introduced "pub_key_tls_client_auth" for the
      Public Key method
   o Added implementation considerations, mainly regarding TLS stack
      configuration and trust chain validation, as well as how to to do
      binding of access tokens to a TLS client certificate for public
      clients, and considerations around certificate bound access tokens
   o Added new section to security considerations on cert spoofing
   o Add text suggesting that a new cnf member be defined in the
      future, if hash function(s) other than SHA-256 need to be used for
      certificate thumbprints

   draft-ietf-oauth-mtls-02

   o Fixed editorial issue https://mailarchive.ietf.org/arch/msg/oauth/
      U46UMEh8XIOQnvXY9pHFq1MKPns
   o Changed the title (hopefully "Mutual TLS Profile for OAuth 2.0" is
      better than "Mutual TLS Profiles for OAuth Clients").

   draft-ietf-oauth-mtls-01

   o Added more explicit details of using RFC 7662 token introspection
      with mutual TLS sender constrained access tokens.
   o Added an IANA OAuth Token Introspection Response Registration
      request for "cnf".
   o Specify that tls_client_auth_subject_dn and
      tls_client_auth_root_dn are RFC 4514 String Representation of
      Distinguished Names.
   o Changed tls_client_auth_issuer_dn to tls_client_auth_root_dn.
   o Changed the text in the Section 3 to not be specific about using a
      hash of the cert.
   o Changed the abbreviated title to 'OAuth Mutual TLS' (previously
      was the acronym MTLSPOC).

   draft-ietf-oauth-mtls-00

   o Created the initial working group version from draft-campbell-
      oauth-mtls

   draft-campbell-oauth-mtls-01

   o Fix some typos.
   o Add to the acknowledgements list.

   draft-campbell-oauth-mtls-00

   o Add a Mutual TLS sender constrained protected resource access
      method and a x5t#S256 cnf method for JWT access tokens (concepts
      taken in part from draft-sakimura-oauth-jpop-04).
   o Fixed "token_endpoint_auth_methods_supported" to
      "token_endpoint_auth_method" for client metadata.
   o Add "tls_client_auth_subject_dn" and "tls_client_auth_issuer_dn"
      client metadata parameters and mention using "jwks_uri" or "jwks".
   o Say that the authentication method is determined by client policy
      regardless of whether the client was dynamically registered or
      statically configured.
   o Expand acknowledgements to those that participated in discussions
      around draft-campbell-oauth-tls-client-auth-00
   o Add Nat Sakimura and Torsten Lodderstedt to the author list.

   draft-campbell-oauth-tls-client-auth-00

   o Initial draft.

Authors' Addresses

Brian Campbell
Ping Identity

Email: brian.d.campbell@gmail.com

John Bradley
Yubico

Email: ve7jtb@ve7jtb.com
URI: http://www.thread-safe.com/

Nat Sakimura
Nomura Research Institute

Email: n-sakimura@nri.co.jp
URI: https://nat.sakimura.org/

Torsten Lodderstedt
YES Europe AG

Email: torsten@lodderstedt.net