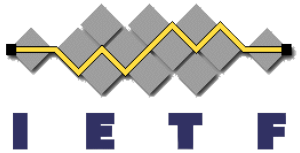
Mutual TLS Profile for OAuth 2.0

Brian Campbell
John Bradley
Nat Sakimura
Torsten Lodderstedt



IETF 100
Singapore
November 2017

draft-ietf-oauth-mtls

https://tools.ietf.org/html/draft-ietf-oauth-mtls-05

What is it?

- Mutual TLS client authentication to the AS
  - Two methods:
    - PKI based
    - Self-signed certificate based mode
- Mutual TLS sender constrained access tokens for protected resources access
  - Certificate bound access tokens

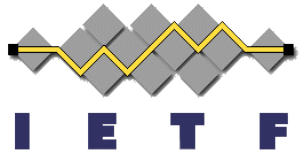
Why?

- Mutual TLS client authentication, which provides better security characteristics than shared secrets, is something that's been done in practice for OAuth but we've never had a spec for it
- Mutual TLS sender constrained resources access binds access tokens to the client certificate so they can't be (re)played or used by any other entity without proof-of-possession
- Banks "need" these for server to server API use cases being driven by new open banking regulations
- Referenced by FAPI's "Read and Write API Security Profile" as a suitable holder of key mechanism
- Referenced by Open Banking API Security Profile

How Mutual TLS Client Authentication Works

- MTLS client authentication to the authorization server
  - TLS connection from client to token endpoint is established with mutual X509 certificate authentication
  - Client includes the "client_id" HTTP request parameter in all requests to the token endpoint
  - AS verifies that the MTLS certificate is the ‘right’ one for the client
  - Metadata supporting the PKI method
    - Authentication Method Name: "tls_client_auth”
    - Client Metadata: "tls_client_auth_subject_dn” specifies the expected subject distinguished name of the client certificate
  - Metadata supporting the Self-signed Certificate method
    - Authentication Method Name: "self_signed_tls_client_auth”
    - The existing "jwks_uri" or "jwks" RFC7591 metadata parameters used to convey a client's certificate(s)

How Mutual TLS Sender Constrained Access Works



- Mutual TLS sender constrained resource access
  - AS associates a hash of the certificate with the access token
    - certificate bound access token
  - TLS connection from client to resource is mutually authenticated TLS
    - The protected resource matches certificate from TLS connection to the certificate hash in the access token
  - JWT Confirmation Method
    - X.509 Certificate SHA-256 Thumbprint Confirmation Method: x5t#S256
  - Confirmation Method for Token Introspection
    - Same data as JWT x5t#S256 confirmation returned in the introspection response and checked by the protected resource
  - Doesn't vary based on client authentication method

JWT Confirmation

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

Token Introspection

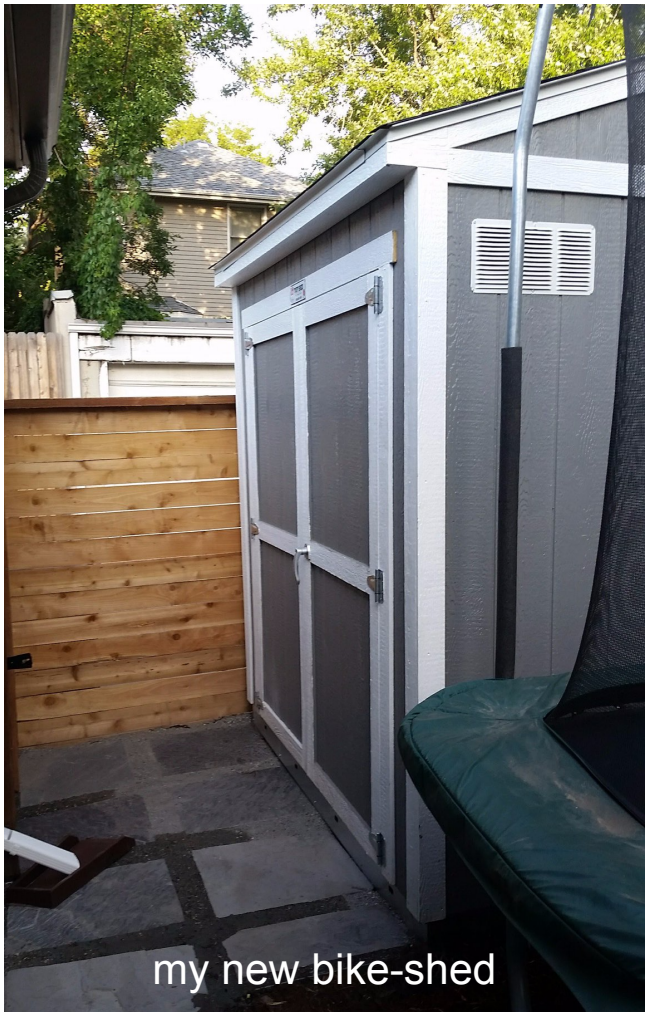
```
HTTP/1.1 200 OK
Content-Type: application/json

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

Recent History

- -02 presented in Prague with somewhat premature call for WGLC
- -03 / -04 / -05
  - Defined two methods of client authentication (binding between certificate and client and trust model)
    - PKI method and Self-Signed Certificate method
  - Introduced AS metadata and client registration parameters to publish and request support for mutual TLS sender constrained access tokens
  - Clarified that MTLS client authentication isn't exclusive to the token endpoint and can be used with other endpoints that utilize client authentication (e.g. revocation and introspection)
  - Removed the "tls_client_auth_root_dn" client metadata field
  - Reorganized doc to more clearly distinguish between:
    - MTLS client authentication and certificate bound access tokens
    - The two client authentication methods

is it bike shedding, bikeshedding or bike-shedding?

- On list comment that the metadata/ registration parameter name in support of MTLS bound ATs was unnatural
  - Current:
    - mutual_tls_sender_constrained_access_tokens
  - Other prospective options:
    - certificate_bound_access_tokens
    - mutual_tls_certificate_constrained_access_tokens
    - mutual_tls_certificate_bound_access_tokens
    - mtls_certificate_bound_access_tokens
    - mtls_bound_access_tokens
    - tls_client_bound_access_tokens
    - etc_etc_etc
  - Opinions are like...



Next Steps...

- Try anew for WGLC so you don't have to listen to this again in London?