UDAP JWT-BASED CLIENT AUTHENTICATION

DRAFT 2018-08-14

Many organizations have established digital identities that can be reused within a trust community to facilitate cross-organizational queries. UDAP JWT-based client authentication leverages existing public key infrastructure and validated identities to help scale such queries.

UDAP implements JWT-based client authentication as an extension to the OAuth 2.0 authorization framework defined in RFC 6749, based in part on the profiles defined in RFC 7521 and RFC 7523 for assertion-based authentication. To request use of this extension, a Client App constructs and digitally signs a JSON Web Token (JWT) that is used by the Client App as an Authentication Token (AnT), then includes this token as a client assertion to authenticate itself to the Authorization Server's token endpoint in its request for an access token. This client authentication protocol can be used with any OAuth 2.0 grant mechanism where a Client App authenticates to the Authorization Server token endpoint in order to obtain an access token, including authorization code flow, client credentials flow, or other extension grant flows utilizing the token endpoint.

Before granting an access token, the Authorization Server uses standard Public Key Infrastructure tools to validate the digital signature on the AnT submitted by the Client App, and to evaluate the trust chain for the Client App's X.509 certificate. The access token is granted by the Authentication Server only if the AnT is valid (see Section 6) and the Client App's certificate is trusted. This protocol SHALL only be used only by Client Apps that are able to protect the private key used to sign AnTs, e.g. confidential clients and certain native device apps.

Note: The HTTP request and response examples presented below are non-normative. For readability purposes, line wraps have been added to some HTTP headers and request bodies, some headers have been omitted, and some parameters have not been URL encoded.

The following steps define the workflow:

1. The Client App checks that the Authorization Server supports UDAP by retrieving the Authorization Server's UDAP metadata from a well-known URL.

```
GET /.well-known/udap HTTP/1.1
Host: resourceholder.example.com
```

Response:

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

{
  "x5c" : ["{cert1}", "{cert2}", …]
}
```

If the Authorization Server returns an error code, then the Authorization Server does not support UDAP JWT-based client authentication and the Client App should abort this workflow. The Authorization Server's certificates are discovered as part of this metadata request. The Client App MAY examine the certificates to determine whether the Authorization Server is part of a suitable trust community.

2. The Client App MUST register with the Authorization Server to use signed JWTs as Authentication Tokens, unless the Client App and Authorization Server support the alternative flow for unregistered client apps described in section 8.1. If the Authorization Server supports UDAP Dynamic Client Registration, the Client App MAY register with the AS using that protocol. If the AS does not support dynamic client registration, it SHOULD supply another method for Client Apps to register their certificates and obtain a client ID.

3. The Client App SHOULD perform any steps required by the grant mechanism it is using before making its request to the token endpoint. These steps occur prior to client authentication.

3.1 Authorization Code Flow

When using the authorization code flow, the Client App should first direct the end user to the AS's authorization endpoint in order to receive an authorization code via the Client App's redirection endpoint, as in this example:


```
GET /authorize?
  response_type=code&
  state_client_random_state&
```

```
  scope=_resource_scope1+resource_scope2&
  redirect_uri=https://client.example.net/clientredirect HTTP/1.1
Host: resourceholder.example.com
```

If the end user authorizes the Client App to the requested resources, the AS will return an authorization code to the Client App by redirecting the user's browser to the Client App's redirection endpoint:

```
HTTP/1.1 302 Found
Location: https:// client.example.net/clientredirect?
   code=authz_code_from_resource_holder&
   state=client_random_state
```

3.2 Client Credentials Flow

When using the client credentials flow, no additional steps are required prior to connecting to the token endpoint.

3.3 For other grant types, perform the steps required by that grant type, if any, prior to connecting to the token endpoint.

4. The Client App then prepares an Authentication Token (AnT) for inclusion in the Client App's request to the AS's token endpoint. The AnT serves two purposes: it establishes the Client's control of a private key, and it provides the digital certificate needed to validate the signature and establish trust. The AnT is a signed JWT containing the following claims:

iss: unique identifying URI of the Client Token Service
sub: the client ID issued by the AS to the Client App
aud: token endpoint URI of the Authorization Server
exp: token expiration time expressed as seconds since epoch (should be short-lived)
iat: issued at time expressed as seconds since epoch
jti: token identifier used to identify token replay

The JOSE Header for the AnT contains the following key/value pairs:

alg : "RS256"
x5c : [cert1, cert2, …]
x5u : valid URI (optional)

The x5c claim contains the Client App's certificate chain as an array of one or more elements, each containing a base64 encoded representation of the DER encoded X.509 certificate. The AnT is digitally signed and assembled using JWS compact serialization as per RFC 7515.

5. The client uses this AnT to authenticate itself as part of its request for an access token from the Authorization Server's token endpoint. The client indicates that an AnT will be used by including two extension parameters: "client_assertion_type" with a value of "urn:ietf:params:oauth:client-assertion-type:jwt-bearer" and "client_assertion" with a value equal to the JWS compact serialization of the signed AnT constructed in step 4. An additional extension parameter "udap" is also included with a string value of "1" to signal to the AS that version 1 of this protocol is being used. The Client App MUST NOT use HTTP Basic authentication, i.e. an Authorization header MUST NOT appear in the request headers, as no shared client secret is used in this workflow. The client ID MAY be included in the request parameter; if included, it must match the sub value in the AnT.

5.1 Authorization Code Flow

Continuing the previous example using the authorization code flow, the Client App submits the authorization code obtained following user authorization:

```
POST /token HTTP/1.1
Host: as.example.com
Content-type: application/x-www-form-urlencoded

grant_type=authorization_code&
  code=authz_code_from_resource_holder&
  client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer&
  client_assertion=eyJh[…remainder of AnT omitted for brevity…]&
  udap=1
```


5.2 Client Credentials Flow

```
POST /token HTTP/1.1
Host: as.example.com
Content-type: application/x-www-form-urlencoded

grant_type=client_credentials&
  client_assertion_type=urn:ietf:params:oauth:client-assertion-type:jwt-bearer&
  client_assertion=eyJh[…remainder of AnT omitted for brevity…]&
  udap=1
```

6. Authorization Server (AS) validates the Client App's request.

6.1 The AS validates the digital signature on the AnT using the public key extracted from cert1 in the x5c parameter of the JOSE header. If the signature cannot be validated, the request is denied.

6.2 The AS attempts to construct a valid certificate chain from the Client's certificate (cert1) to an anchor certificate trusted by the AS using conventional X.509 chain building techniques and path validation, including certificate validity and revocation status checking. The Client MUST include its own certificate and MAY include a complete certificate chain in its request. The AS MAY use additional certificates not included by the Client to construct a chain (e.g. from its own certificate cache or discovered via the X.509 AIA mechanism). If a trusted chain cannot be built and validated by the AS, the request is denied.

6.3 The AS validates the sub, aud, exp, iat, and jti values within the AnT. The sub value MUST correspond to a registered client ID that is permitted to authenticate using an AnT. If the request contains a client_id parameter, the client_id value MUST match the sub value. The aud value MUST contain the AS's base URL, and the AnT MUST be unexpired. A maximum AnT lifetime of 5 minutes is RECOMMENDED. The AS MAY deny a request if the same AnT (as determined by the jti value) has been used in a previous token request.

6.4 The AS validates any other parameters in the request as per the requirements of the grant mechanism identified by the grant_type value. If a parameter is invalid or a required parameter is missing, the request is denied.

6.5 The AS MAY apply additional authorization constraints based on certificate attributes as a matter of local policy.

7. Authorization Server responds to request

7.1 If the request is approved, the Authorization Server returns a token response as per Section 5.1 of RFC 6749. For example:

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

{
   "access_token": "example_access_token_issued_by_AS",
   "token_type": "Bearer",
   "expires_in": 3600
}
```

The AS MAY also return a refresh_token in its response.

7.2 If the request is denied, the AS returns an error as per Section 5.2 of RFC 6749. Denials related to trust validation SHOULD use the "invalid_client" code. Denials related to invalid signatures should use the "invalid_request" code. The AS MAY include an error_description parameter. For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/json

{
   "error": "invalid_client",
   "error_description": "The submitted authentication token has expired"
}
```

8 Client Authentication without Pre-registration

8.1 When the identity and privileges of a Client App can be fully determined by an Authorization Server based solely on the attributes listed in the client certificate included in the Authentication Token, the Authorization Server MAY allow an unregistered Client App to use this authentication protocol to obtain an access token using the client_credentials grant flow. In this case, as no client_id exists for the unregistered Client App, the corresponding trust community MUST define the set of certificate attributes that an AS can use to uniquely identify a Client App, and specify a reserved value for the sub claim (e.g. "unregistered")

that can be included in the AnT to signal to the AS that the subject of the AnT is the unregistered Client App identified by the included certificate. This specification does not restrict how the Authorization Server and/or Resource Server communicate to the Client App that this alternate workflow can be used.

Note that this approach is not suitable for applications utilizing the authorization code flow as there is no mechanism to preregister the Client App's redirection URI; such unregistered clients SHOULD use the UDAP Dynamic Client Registration Protocol instead.

9 References

Campbell, C., et al. "Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7521, RFC Editor, May 2015.
Cooper, D., et al. "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, RFC Editor, May 2008.
Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, RFC Editor, October 2012.
Jones, M., et al. "JSON Web Signature (JWS)", RFC 7515, RFC Editor, May 2015.
Jones, M., et al. "JSON Web Token (JWT)", RFC 7519, RFC Editor, May 2015.
Jones, M., et al. "JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants", RFC 7523, RFC Editor, May 2015.
Sakimura, N, et al. "OpenID Connect Core 1.0 incorporating errata set 1", The OpenID Foundation, November 2014.

10 Authors

Luis C. Maas III, EMR Direct
Julie W. Maas, EMR Direct

11 Notices

ABOUT UDAP

The Unified Data Access Profiles (UDAP) published by UDAP.org increase confidence in open API transactions through the use of trusted identities and verified attributes. Interest in UDAP led to the development of additional implementation guides focused on key use cases in the deployment of reusable identities, including Dynamic Client Registration and Tiered OAuth. The profiles can be used to help scale the secure use of open APIs, while also protecting the personal information of network participants.

CHECK IN...

✉ Email collaborate@udap.org for more information or
Join our Google Group (https://groups.google.com/forum/#!forum/udap-discuss/join)
to participate in the development of UDAP.

GETTING INVOLVED

UDAP profiles have been tested at several HL7 FHIR connectathons. Contact us for more information about how to become involved in testing or to pilot one or more profiles within your own ecosystem.

Interoperability Engine (http://www.interopengine.com) OAuth and FHIR services support UDAP.

HealthToGo (https://www.healthtogo.me) FHIR Client services support UDAP.

UDAP SPONSORS

The development of these profiles is currently sponsored by EMR Direct. Visit the EMR Direct (http://www.emrdirect.com) corporate site for more information on integrating interoperability services into software applications, or for activating production Direct Messaging or HL7 FHIR services.

(https://www.twitter.com/udapTools)