Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 16, 2019

J. Richer, Ed.
Bespoke Engineering
May 15, 2019

Transactional Authorization
draft-richer-transactional-authz-00

Abstract

This document defines a mechanism for delegating authorization to a piece of software, and conveying that delegation to the software.

Requirements Language

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 BCP 14 RFC 2119 [RFC2119] RFC 8174 [RFC8174] when, and only when, they appear in all capitals, as shown here.

Status of This Memo

Copyright Notice

Table of Contents

1. Parties

The Authorization Server (AS) manages the transactions. It is defined by its transaction endpoint, a single URL that accepts a POST request with a JSON payload. The AS can also have other endpoints, including interaction endpoints.

The Authorization Requester (AR) is a party calling the AS. It can be acting as either RC or RS. (TODO: there needs to be a better term for this.)

The Resource Client (RC) requests tokens from the AS and uses tokens at the RS.

The Resource Server (RS) accepts tokens from the RC and validates them (potentially at the AS).

The Resource Owner (RO) authorizes the request from the RC to the RS

2. Transaction request

To start a transaction, the RC makes a transaction request to the transaction endpoint of the AS. The RC creates a JSON document with up to five sections.

client Information about the RC making the request, including display name, home page, logo, and other user-facing information. This section is RECOMMENDED.

resources Information about the RS's the resulting token will be applied to, including locations, extents of access, types of data being accessed, and other API information. This section is REQUIRED.

user Information about the RO as known to or provided to the RC, in
   the form of assertions or references to external data. This
   section is OPTIONAL.

interact Information about how the RC is able to interact with the
   RO, including callback URI's and state. This section is REQUIRED
   if the client is capable of driving interaction with the user.

keys Information about the keys known to the RC and able to be
   presented in future parts of the transaction. This section is
   REQUIRED. (Note: I can't think of a good reason for this to be
   optional.)

An AS MAY

2.1. Client

This section provides descriptive details of the client software
making the call.

name Display name of the client software

uri User-facing web page of the client software

logo_uri Display image to represent the client software

```
client: {

  name: "Display Name",
  uri: "https://example.com/client"

}
```

This can also be presented as a client handle reference.

2.2. Resource

This section identifies the RS and describes what the RC wants to do
with the API hosted at the RS. This section is an array of objects,
each object representing a single resource set. That AS MUST
interpret the request as being for all of the resources listed.

actions The types of actions the RC will take at the RS

locations URIs the RC will call at the RS

data types of data available to the RC at the RS's API

```
resources: [
  {
    actions: ["read", "write"],
    locations: ["https://exapmle.com/resource"]
    data: ["foo", "bar"]

  }
]
```

This can also be presented as a set of resource handle references.

2.3. User

This section provides a verifiable assertion about the RO interacting with the client on behalf of the request.

assertion The value of the assertion as a string.

type The type of the assertion. Possible values include "oidc_id_token"...

```
user: {

  assertion: "eyj0....",
  type: "oidc_id_token"

}
```

This can also be presented as a user handle reference.

2.4. Interact

This section provides details of how the RC can interact with the RO.

}

All interact requests MUST have the "type" field.

type REQUIRED. Type of interaction. Can be "redirect" or "device".

Each interaction type has its own parameters and behaviors, detailed below.

This can also be presented as interaction handle reference.

2.4.1. Redirect

A redirect type interaction has the RC send the RO to a URL at the AS and interact with the AS directly, using any number of interactions. Following the interaction, the RO is sent back to the RC using the "callback" URI.

type MUST be "redirect"

callback REQUIRED. URI to send the user to after interaction, SHOULD (MUST?) be unique per transaction and hosted or accessible by the RC. This URL MUST NOT contain any fragment component. This URL MUST be protected by HTTPS, hosted on a server local to the user's browser ("localhost"), or use an application-specific URL scheme. MAY be limited by the AS based on the client's information.

state REQUIRED. Unique value to be returned to the application as a query parameter on the callback URL, must be sufficiently random to be unguessable by an attacker. MUST be generated by the client for this transaction.

```
interact: {
  type: redirect
  callback: https://client.foo/
  state: foo
}
```

2.4.2. Device

The device type interaction has the RC instruct the user to go to a URL at the AS using a secondary device. The user then interacts with the AS directly by entering a short code provided by the AS to the RC. Following the interaction, the RO is prompted by the AS to check their RC device, which can poll the AS until the authorization is complete.

type MUST be "device"

```
interact: {
  type: device
}
```

2.5. Keys

This section lists the keys that the client can present proof of ownership. Each key type has its own proofing mechanism and additional required parameters, listed in individual sections below.

type Validation method for the key, must be one of "jwsd", "mtls/x509", or "did/zkp".

All presented keys MUST be validated by the AS as per the Key Validation section.

This can also be presented as a key handle reference. The key referenced by a handle MUST be validated by the AS.

```
key: {

  handle: "3eru876tyhgr5678ikjhgt"

}
```

2.5.1. Detatched JWS method

type MUST be "jwsd"

jwks Value of the public key as a JWK Set JSON object [Note: should this be a single JWK instead? And do we want to bother with url-based references?]. MUST contain an "alg" field which is used to validate the signature. MUST contain the "kid" field to identify the key in the signed object.

```
   key: {

     type: jwsd,
     jwks: { keys: [ alg: RS256, kty: ... ] }

   }
```

2.5.2. MTLS method

type MUST be "mtls"

cert REQUIRED. String serialized value of the certificate thumbprint as per OAuth-MTLS.

```
   key: {

     type: mtls,
     cert: "MII...."

   }
```

2.5.3. DID method

type MUST be "did"

did The DID URL identifying the key (or keys) used to sign this request.

```
   key: {

     type: did,
     did: "did:v:foo...."

   }
```

3. Interaction response

When evaluating a transaction request, the AS can determine that it needs to have the RO present to interact with the AS before issuing a token. This interaction can include the RO logging in to the AS, authorizing the transaction, providing proof claims, determining if the transaction decision should be remembered for the future, and other items.

The AS responds to the RC based on the type of interaction supported by the RC in the transaction request.

This response can indicate a set of keys are bound to the transaction as in Key Binding. This response includes a transaction handle as in Transaction Handle.

3.1. Redirect interaction

If the RC supports a "redirect" style interaction, the AS creates a unique interaction URL and returns it to the RC. This URL MUST be associated with a single pending transaction.

interaction_url The interaction URL that the RC will direct the RO to. This URL MUST be unique to this transaction request. The URL SHOULD contain a random portion of sufficient entropy so as not to be guessable by the user. The URL MUST NOT contain the transaction handle or any client identifying information. This URL MUST be protected by HTTPS. This URL MUST NOT contain any fragment component.

handle The transaction handle to use in the continue request once the RO has been returned to the RC via the callback URL. See the section on transaction handles.

```
{

  interaction_url: "https://server.example.com/interact/123asdfklj",
  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    method: "bearer"
  }
}
```

When the RC receives this response, it sends the RO to the interaction URL. When interacting with the RO, the AS MAY perform any of the behaviors in the User Interaction section.

Once the RO has completed the interaction with the AS, the AS returns the user to the RC by redirecting the RO's browser to the RC's callback URL presented at the start of the transaction, with the state parameter appended to the callback URL as a query parameter in addition to an interaction handle to be returned to the AS in a transaction continuation request.

state REQUIRED. The (hashed?) value of the state parameter sent by the client in the initial interaction request.

interact_handle REQUIRED. A shared secret associated with this interaction. This value MUST be sufficiently random so as not to be guessable by an attacker. This value MUST be associated by the AS with the underlying transaction that is associated to with this interaction.

Upon processing this request to the callback URL, the client MUST match the state value to the value it sent in the original transaction request. The RC then sends a transaction continuation request with the transaction handle returned in the interaction response and the (hash of?) the interaction handle returned as a query parameter to the callback URL.

The client sends the hash of the interaction handle as the "interact_handle" field of the transaction continuation request.

```
{
    "handle": "80UPRY5NM33OMUKMKSKU",
    "interact_handle": "CuD9MrpSXVKvvI6dN1awtNLx-HhZy46hJFDBicG4KoZaCmBofvqPxtm
}
```

[Open Question: error conditions. If the user denies access or there's some other authorization error, do we return to the callback? What's the attack surface here? We could always return an error page to the browser and cancel the underlying transaction, effectively killing it at the AS.]

If the AS cannot identify the source transaction from the source URL, it returns an HTTP 404 error page to the browser and optionally an error message to the user.

3.2. Secondary device interaction

If the RC supports a "device" style interaction, the AS creates a unique interaction code and returns it to the RC along with a URL to give the user for interaction.

user_code A short code that the user can type into an authorization server. This string MUST be case-insensitive, MUST consist of only easily typeable characters (such as letters or numbers). The time in which this code will be accepted MUST be short lived.

interaction_url The interaction URL that the RC will direct the RO to. This URL SHOULD be stable over time.

wait The amount of time to wait before polling again, in integer seconds.

handle The transaction handle to use in the continue request. See the section on transaction handles.

```
{

  user_code: "ABCD1234"
  interaction_url: "https://server.example.com/device",
  wait: 30,
  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    method: "bearer"
  }
}
```

When the RC receives this response, it MUST communicate the user code to the RO. If possible the RC SHOULD communicate the interaction URL to the user as well, although this can be a stable URL at the AS.

4. Wait response

If the AS needs to do something that the RC has no part in before it can give a definitive response, the AS replies to the transaction request with a wait response. This tells the RC that it can poll the transaction after a set amount of time.

This response can indicate a set of keys are bound to the transaction as in Key Binding. This response includes a transaction handle as in Transaction Handle.

wait REQUIRED. The amount of time to wait before polling again, in integer seconds.

handle REQUIRED. The transaction handle to use in the continue request. This MUST be a newly-created handle and MUST replace any existing handle for this transaction. See the section on transaction handles.

```
   {

     wait: 30,
     handle: {
       value: "tghji76ytghj9876tghjko987yh",
       method: "bearer"
     }

   }
```

5. Interaction at the AS

When the RO is interacting with the AS at the interaction endpoint, the AS MAY perform whatever actions it sees

6. Error response

If the AS determines that the token cannot be issued for any reason, it responds to the client with an error message. This message does not include a transaction handle, and the RC can no longer poll for this transaction. The RC MAY create a new transaction and start again.

error The error code.

```
   {

     error: user_denied

   }
```

TODO: we should have a robust error mechanism.

7. Transaction continue request

Once a transaction has begun, the AS associates that transaction with a transaction handle which is returned to the RC in one of the transaction responses. This handle MUST be unique, MUST be associated with a single transaction, and MUST be one time use.

The RC continues the transaction by making a request with the transaction handle in the body of the request. The RC MAY add additional fields depending on the type of interaction and authorization process in play.

transaction The (hash of?) transaction handle to use in the continue request.

interaction_handle The (hash of?) interaction handle returned to the RC's callback URL from the interaction endpoint.

```
{

  transaction: "tghji76ytghj9876tghjko987yh"

}
```

8. Token response

access_token The access token that the RC uses to call the RS.

access_token_keys List of keys that the access token is bound to using the methods in key validation. If not specified, the access token is a bearer token.

handle The transaction handle to use in the continue request to get a new access token once the one issued is no longer usable. See the section on transaction handles.

```
key: {

  access_token: "08ur4kahfga09u23rnkjasdf",
  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    method: "bearer"
  }

}
```

9. Handle references

Many parts of this protocol are referenced through the use of handles as stand-ins for actual values, including transactions themselves as well as portions of transactions.

value The value of the handle as a string.

method The verification method, MUST be one of "bearer" or "sha3".

9.1. Validating handles

Bearer handles are validated by doing an exact byte comparison of the string representation of the handle value.

SHA3 handles are validated by taking the SHA3 hash of the handle value and encoding it in Base64URL with no padding.

9.2. Transaction handles

Transaction handles are issued by the AS to the RC to allow the RC to continue a transaction after every step. A transaction handle MUST be discarded after it is used. If the AS determines that the RC can continue the transaction, a new transaction handle will be issued in its place.

9.3. Client handles

Client handles stand in for the client section of the initial transaction request. The AS MAY issue a client handle to a client as part of a static registration process, analogous to a client ID, allowing the client to be associated with an AS-side configuration that does not change at runtime. Such static processes SHOULD be bound to a set of keys known only to the client software.

Client handles MAY be issued by the RS in response to a transaction request. The AS MAY bind this handle to the interact, resource, and key handles issued in the same response. When the RC receives this handle, it MAY present the handle in future transaction requests instead of sending its information again.

```
{

  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    method: "bearer"
  },
  client_handle: {
    value: "absc2948afgdkjnasdf9082ur3kjasdfasdf89",
    method: "bearer"
  }

}
```

The RC sends its handle in lieu of the client block of the transaction request:

```
{

  client: {
    handle: "absc2948afgdkjnasdf9082ur3kjasdfasdf89"
  }

}
```

9.4. Resource handles

Resource handles stand in for the detailed resource request in the transaction request. Resource handles MAY be created by the authorization server as static stand-ins for specific resource requests, analogous to OAuth2 scopes.

Resource handles MAY be issued by the RS in response to a transaction request. When the RC receives this handle, it MAY present the handle in future transaction requests instead of sending its information again.

```
   {

     handle: {
       value: "tghji76ytghj9876tghjko987yh",
       method: "bearer"
     },
     resource_handle: {
       value: "foo",
       method: "bearer"
     }

   }
```

The RC sends its handle in lieu of the resource block of the transaction request:

```
   {

     resource: {
       handle: "foo"
     }

   }
```

9.5. User handles

User handles MAY be issued by the AS in response to validating a specific RO during a transaction. This handle can be used in future transactions to represent the current user, analogous to the persistent claims token.

```
   {

     handle: {
       value: "tghji76ytghj9876tghjko987yh",
       method: "bearer"
     },
     user_handle: {
       value: "absc2948afgdkjnasdf9082ur3kjasdfasdf89",
       method: "bearer"
     }

   }
```

The RC sends its handle in lieu of the user block of the transaction request:

```
  {

    user: {
      handle: "absc2948afgdkjnasdf9082ur3kjasdfasdf89"
    }

  }
```

9.6. Key handles

Key handles stand in for the keys section of the initial transaction request. The AS MAY issue a key handle to a client as part of a static registration process, allowing the client to be associated with an AS-side configuration that does not change at runtime.

Key handles MAY be issued by the RS in response to a transaction request. The AS SHOULD bind this handle to the client, resource, and user handles issued in the same response. When the RC receives this handle, it MAY present the handle in future transaction requests instead of sending its information again.

```
{

  handle: {
    value: "tghji76ytghj9876tghjko987yh",
    method: "bearer"
  },
  key_handle: {
    value: "absc2948afgdkjnasdf9082ur3kjasdfasdf89",
    method: "bearer"
  }

}
```

The RC sends its handle in lieu of the client block of the transaction request:

```
{

  key: {
    handle: "absc2948afgdkjnasdf9082ur3kjasdfasdf89"
  }

}
```

When the AS receives a key handle, it MUST validate that the keys referenced by the handle are bound to the current transaction request.

10. Binding Keys

Any keys presented by the RC to the AS or RS MUST be validated as part of the transaction in which they are presented. Any keys bound to the transaction are indicated by the bound_keys section of the transaction response. Any keys referenced in this section MUST be used with all future transaction requests.

10.1. Binding a key to a transaction

Keys are bound to a transaction by including a bound_keys field in the transaction response alongside the transaction handle. Any further keys used for binding

10.2. Validating detached JWS

To sign a request to the transaction endpoint, the RC takes the serialized body of the request and signs it using detached JWS. The header of the JWS MUST contain the kid field of the key bound to this client during this transaction. The header MUST contain an alg field appropriate for the key identified by kid and MUST NOT be none. The

The RC presents the signature in the JWS-Signature HTTP Header field. [Note: this is a custom header field, do we need this?]

```
JWS-Signature: eyj0....
```

When the AS receives the JWS-Signature header, it MUST parse its contents as a detached JWS object. The HTTP Body is used as the payload for purposes of validating the JWS, with no transformations.

10.3. Validating attached JWS

[Note: if we do an attached JWS we end up having two different data types to deal with at the AS, is this ok?]

To sign a request to the transaction endpoint with an attached JWS, the RC takes the body of the request as the JWS payload and wraps the request in a JWS object.

10.4. Validating MTLS

The RC presents its client certificate during TLS negotiation with the server. The AS or RS takes the thumbprint of the client certificate presented during mutual TLS negotiation and compares that thumbprint to the thumbprint presented by the RC application.

10.5. Validating DID

The RC signs the request using [some HTTP signing mechanism] and its private key, and attaches the signature to the HTTP request using [a header method?]. [Note: is DID just a key-lookup mechanism here or should we use a different kind of crypto method as well?]

11. Using a Token

Bearer access tokens issued through this method can be used with the authorization header method found in RFC6750. Other access tokens are validated by the RS in accordance with the methods in the Binding

Keys section.

12. Acknowledgements

13. IANA Considerations

This specification creates one registry and registers several values into existing registries.

14. Security Considerations

15. Privacy Considerations

16. Normative References

[OpenID] Sakimura, N., Bradley, J., and M. Jones, "OpenID Connect Core 1.0", November 2014.

[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>.

[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>.

[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>.

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

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26,

              RFC 8126, DOI 10.17487/RFC8126, June 2017,
              <https://www.rfc-editor.org/info/rfc8126>.

   [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC
              2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174,
              May 2017, <https://www.rfc-editor.org/info/rfc8174>.

   [RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data
              Interchange Format", STD 90, RFC 8259,
              DOI 10.17487/RFC8259, December 2017,
              <https://www.rfc-editor.org/info/rfc8259>.

Appendix A. Document History

   - 00

   o Initial submission.

Author's Address

   Justin Richer (editor)
   Bespoke Engineering

   Email: ietf@justin.richer.org