IETF 108 GNAP WG

Justin Richer

GNAP
GNAP!

https://oauth.xyz/

- Detailed examples of proposed protocol
- Test implementations in Java, NodeJS, and React
- Individual draft specification

What’s new in -09?

- Refactored based on request and response parts
- Management URLs for grants and tokens
- Refocus claims requests on subject identifiers
- Interaction capabilities for apps, pushback
- Alignment with RAR in resource structure
- Simplified dynamically returned handles

What's not new?

- Polymorphic JSON and passing by reference
- Clients identified by keys
- Resources use rich structure with possible shortcuts
- Interaction uses inline negotiation
- Single and multiple access tokens
- Compatibility with OAuth 2 constructs

Document Structure

Section 2: Request

- Everything a client sends to start the process
- New section on interaction requests
  - Same in-line negotiation protocol, just all together now
- Clarified polymorphism inline with use
  - Same process, just not in a different section
- Clarified requesting resources and user information
  - Align with RAR and secevent-subject-identifiers
- Clarified presenting user information

Section 3: Response

- Everything the AS can respond with
  - Access tokens (single and multiple)
  - Interaction methods (next-steps in negotiation)
  - User information (aligned with secevent-subject-ids)
- Continuation and token management have URIs
  - These could be stable or dynamic, depending on AS
  - Client always takes value as given

Section 4: Interaction at the AS

- How the AS has to be prepared to deal with the responses to interaction in Section 3
- Requirements for getting to the AS and getting back are separated from each other
  – Common flows will mix them together as needed

Section 5: Continuing a Request

- Follow-on from the "continue" response of Section 3
  - Possibly augmented with information from Section 4.4
- Similar to initial request/response
  - Can have some of the same inputs (Section 2)
  - Can have most of the same outputs (Section 3)
- Could keep going after tokens are issued … maybe?

Section 6: Managing Tokens

- If token management URL is given, client can rotate and/or revoke using this URL
- Token is used to access the management URL
  - Always bound with keys (either client's or token's)

Section 7: Sending Tokens

- Bearer tokens: just use RFC6750
- Other tokens: use key bindings from Section 8
- Potentially a separate document (but it's really short)

Section 8: Binding Keys

- General-purpose methods for tying a key to a request within the GNAP protocol
- Could be requests between different parties:
  - Client->AS
  - Client->RS
  - RS->AS
- Probably too many methods here but all have implementations

Section 9: Discovery

- Protocol is designed to allow in-line negotiation of nearly all options and elements
- Pre-request discovery allowed for clients that programmatically optimize things
  - But not required for functionality

Section 10: Resource Servers

- Token introspection
- Downstream RS-to-RS chaining
- How to start a Client-to-RS-first protocol
- Probably should be in separate documents, but there's interest in the community for this
  - It's really unbaked

Appendix B: Data Models

- OAuth 2 is lacking internally consistent data models
- We need to define common structure for:
  - AS (facilitate discovery and deployment decisions)
  - Client (facilitate registration and interaction assumptions)
  - RS (facilitate discovery and downstream processing)
  - Token/access rights (facilitate token formats and introspection)
- If we don't do this now, it'll get back-patched in future specs like OAuth 2

Appendix C: Examples

- Now that the draft is cut into message pieces, full protocol examples are moved to the appendix
- Currently includes Auth Code, Device, Client Credentials, Asynchronous, and OAuth 2 examples

Details!

Continuation Structure

Continuing a request

AS:

```
{
  "continue": {
    "handle": "80UPRY5NM33OMUKMKSKU",
    "uri": "https://server/continue",
    "wait": 60
  }
}
```

Client (to https://server/continue):

```
{
 "handle": "80UPRY5NM33OMUKMKSKU"
}
```

Stable but separate from initial URL

Continuing a request (alternate)

AS:

```
{
  "continue": {
    "handle": "80UPRY5NM33OMUKMKSKU",
    "uri":
"https://server/continue/80UPRY5N",
    "wait": 60
  }
}
```

Client (to https://server/continue/80UPRY5N):

```
{
 "handle": "80UPRY5NM33OMUKMKSKU"
}
```

Unique to request

Continuing a request (alternate 2)

AS:

```
{
  "continue": {
    "handle": "80UPRY5NM33OMUKMKSKU",
    "uri": "https://server/tx",
    "wait": 60
  }
}
```

Client (to https://server/tx):

```
{
 "handle": "80UPRY5NM33OMUKMKSKU"
}
```

Same as the initial request URL

Continuing a request after interaction

AS:

```
{
  "continue": {
    "handle": "80UPRY5NM33OMUKMKSKU",
    "uri": "https://server/continue"
  }
}
```

Client (to https://server/continue):

```
{
 "handle": "80UPRY5NM33OMUKMKSKU",
 "interact_ref": "4WIKYBC2PQ6U56NL1"
}
```

Allowing challenge/response interaction

AS:

```
{
  "challenge": {
     "value": "OFXG4Y5CV",
     "origin": "https://server/",
     "alg": "SHA256"
  },
  "continue": {
    "handle": "80UPRY5NM33OMUKMKSKU",
    "uri": "https://server/continue"
  }
}
```

Client (to https://server/continue):

```
{
 "handle": "80UPRY5NM33OMUKMKSKU",
 "challenge_response": {
    "key_id": "2-230-235123",
    "value": "ZXYQNEOAF-32f2/afa="
 }
}
```

Allowing additional continues:

Client (to https://server/continue):

```
{
 "handle": "80UPRY5NM33OMUKMKSKU"
}
```

AS:

```
{
  "continue": {
    "handle": "4IFWWIKYBC2PQ6U56NL1",
    "uri": "https://server/continue",
    "wait": 60
  }
}
```

Rotate the reference and possibly the URI too, client uses what comes back

Why separate URL from reference?

- Allow AS to rotate reference on use for security
  - Considered good practice with refresh tokens today
  - Required in UMA2 equivalent "permission ticket"
- Allow different AS deployments
  - AS can allow stable URLs or dynamic where needed
  - We shouldn't dictate URLs where possible
- Allow reference in derived requests
  - Upscoping, downscoping, token exchange, grant-level refresh and management

Open question: Make reference an access token itself?

- Should we re-use access token semantics and structures to manage the grant itself?
- Similarities:
  - Limited to only grant management API
  - Bound to client key (could be explicit, never bearer)
  - Similar to "directed tokens" discussion
- Vaguely like OAuth 1's "request token"

Token Management

Client Managing Access Tokens

- Client given URL to rotate and revoke token
- Somewhat RESTful API
  - POST to rotate
  - DELETE to revoke
- Token used to access its management API
  - Requires proof of token-bound key if present
  - Requires proof of client-bound key if bearer

Getting the management URL

```
"access_token": {
  "value": "OBW7OZB8CDFONP219RP1LT0",
  "proof": "bearer",
  "manage": "https://server/token"
}
```

Getting the management URL (alt)

```
"access_token": {
  "value": "OBW7OZB8CDFONP219RP1LT0",
  "proof": "bearer",
  "manage": "https://server/token/NP219RP1L"
}
```

Why a separate URI?

- Supporting multiple access tokens pushes to separating concerns from overall "request" and resulting access
- AS can use a stable URL to open firewalls etc.
- Client should already know how to present an access token and bind a key

Interaction Negotiation

Front-channel Binding (Auth Code)

Client:

```
{
 "interact": {
   "redirect": true,
   "callback": {
     "uri": "https://client.foo",
     "nonce": "VJLO6A4CAYLBXHTR0KRO”
    }
  }
}
```

AS:

```
{
   "interaction_url":
"https://server/i/4CF492MLVMSMKMXKHQ",
   "callback_server_nonce": "OFXG4YLH",
   "continue": {
     "handle": "80UPRY5NM33OMUKMKSKU",
     "uri": "https://server/continue",
   }
}
```

User code (Device)

Client:

```
{
 "interact": {
   "user_code": true
 }
}
```

AS:

```
{
   "user_code": {
     "url": "https://server/device",
     "code": "A1BC-3DFF"
   },
   "continue": {
     "handle": "80UPRY5NM33OMUKMKSKU",
     "uri": "https://server/continue",
     "wait": 60
    }
}
```

Allow short URIs?

Client:

```
{
 "interact": {
    "redirect": true,
    "short_redirect": true,
    "user_code": true
 }
}
```

AS:

```
{
  "interaction_url":
"https://server/i/4CF492MLVMSMKMXKHQ",
  "short_interaction_url":
"https://srv.ex/MXKHQ”,
  "user_code": {
    "url": "https://server/device",
    "code": "A1BC-3DFF"
  }
}
```

Allow short URIs?

Client:

```
{
 "interact": {
    "redirect": true,
    "short_redirect": true,
    "user_code": true
 }
}
```

AS:

```
{
  "interaction_url":
"https://server/i/4CF492MLVMSMKMXKHQ",
  "user_code": {
    "url": "https://server/device",
    "code": "A1BC-3DFF"
  }
}
```

Allow short URIs?

Client:

```
{
 "interact": {
    "redirect": true,
    "short_redirect": true,
    "user_code": true
 }
}
```

AS:

```
{
  "short_interaction_url":
"https://srv.ex/MXKHQ”,
  "user_code": {
    "url": "https://server/device",
    "code": "A1BC-3DFF"
  }
}
```

Application URI

Client:

```
{
 "interact": {
   "redirect": true,
   "app": true,
   "callback": {
     "uri": "https://client.foo",
     "nonce": "VJLO6A4CAYLBXHTR0KRO”
    }
  }
}
```

AS:

```
{
  "interaction_url":
"https://server/i/4CF492MLVMSMKMXKHQ",
  "app_url":
"https://app.ex/launch?tx=4CF492MLV"
  "server_nonce": "0FXG4Y5CVJCX821LH",
  "continue": { ... }
}
```

Why a separate app URL?

- AS could want different URIs for captured apps and web-based interaction
- Leave room for additional fields
  - Distributed storage address for drop-off protocols
  - Keys and pointers for onion routing

Client pushback

Client:

```
{
 "interact": {
   "redirect": true,
   "pushback": {
     "uri": "https://client.foo",
     "nonce": "VJLO6A4CAYLBXHTR0KRO”
    }
  }
}
```

AS:

```
{
   "interaction_url":
"https://server/i/4CF492MLVMSMKMXKHQ",
   "pushback_server_nonce": "0FXG4Y5H",
   "continue": {
     "handle": "80UPRY5NM33OMUKMKSKU",
     "uri": "https://server/continue",
   }
}
```

AS push to Client

```
POST /push/554321 HTTP/1.1
Host: client.example.net
Content-Type: application/json

{
   "hash":
"p28jsq0Y2KK3WS__a42tavNC64ldGTBroywsWxT4md_jZQ1R2HZT8
BOWYHcLmObM7XHPAdJzTZMtKBsaraJ64A",
   "interact_ref": "4IFWWIKYBC2PQ6U56NL1"
}
```

Why support push?

- “Callback” assumes user in a browser at the client
- “Pushback” assumes direct connection from AS
  - User is on secondary device
  - Client has connected backend

Why separate redirect and callback?

- Flexible combinations for different use cases
- Client knows what it's capable of
- AS knows what it will allow for a given request

Extend Interaction Safely

Client:

```
{
 "interact": {
    "webauthn": true,
    "didcomm_query": true,
    "app": true,
    "backchannel_push": true,
    ...
 }
}
```

AS:

```
{
    "webauthn": {
        "origin": "server.example",
        "challenge": "A1BC352DFD"
    },
    "app_url": "app:/xyz"
}
```

Open Question: Align Response?

Client:

```
{
 "interact": {
   "redirect": true,
   "callback": {
     "uri": "https://client.foo",
     "nonce": "VJLO6A4CAYLBXHTR0KRO”
    }
  }
}
```

AS:

```
{
   "interaction_url":
"https://server/i/4CF492MLVMSMKMXKHQ",
   "callback_server_nonce": "OFXG4YLH"
}
```

```
{
   "interact": {
     "redirect":
"https://server/i/4CF492MLVMSMKMXKHQ",
     "callback": "OFXG4YLH"
   }
}
```

Identity and User Information

User information directly to the client

Client:

```
{
  "subject": {
    "sub_ids": ["iss-sub", "email"],
    "assertions": ["oidc_id_token"]
  }
}
```

AS:

```
{
  "subject": {
    "sub-ids": [
      { "subject_type": "email",
        "email": "user@example.com”
      }
    ],
    "assertions": {
       "oidc_id_token": "eyj0...”
    }
  }
}
```

Sending information about the user to the AS

```
{
  "user": {
    "sub-ids": [
      { "subject_type": "email",
        "email": "user@example.com”
      }
    ],
    "assertions": {
       "oidc_id_token": "eyj0...”
    }
  }
}
```

Untrusted identifiers

Verifiable assertions

Why only identifiers and assertions?

- Privacy-first design principles
  - Client doesn't know who the user is before calling AS
  - Client doesn't know what information it needs to ask for
- If client does know the user, it's not asking the AS
- Identity schema are complex
  - Better left to dedicated extensions
  - OpenID-GNAP?

Open Questions

- Request as "subject" and declaration as "user"
  - Terms are confusing, better names?
  - "Claims" could come back as a "resource"
- Allow additional items in response?
  - Or other non-claim direct data responses?

Tokens and Resources

@justin _ _ richer

https://bspk.io/

Requesting complex data

Client:

```
{
 "resources": [
   {
    "type": "photo-api",
    "actions": [ "read", "write",
      "dolphin" ],
    "locations": [ "https://server.example.net/",
"https://resource.local/other" ],
    "datatypes": [ "metadata", "images" ]
   },
   {
    "type": "financial-transaction",
    "actions": [ "withdraw" ],
    "identifier": "account-14-32-32-3",
    "currency": "USD”
   }
 ]
}
```

AS:

```
{
 "access_token": {
   "value": "MHKUR64TB8N6BW7OZB8CDFONP219RP1LT0",
   "proof": "bearer",
   "resources": "resources": [
   {
    "type": "photo-api",
    "actions": [ "read", "write",
      "dolphin" ],
    "locations": [ "https://server.example.net/",
"https://resource.local/other" ],
    "datatypes": [ "metadata", "images" ]
   },
   {
    "type": "financial-transaction",
    "actions": [ "withdraw" ],
    "identifier": "account-14-32-32-3",
    "currency": "USD”
   }
  ]
 }
}
```

Requesting predefined data structures

Client:

```
{
 "resources": [
     "read", "dolphin-metadata",
     "some other thing"
 ]
}
```

AS:

```
{
 "access_token": {
   "value": "MHKUR64TB8N6BW7OZB8T0",
   "proof": "bearer",
   "resources": [
     "read", "dolphin-metadata",
     "some other thing"
   ]
  }
}
```

Equivalence between items

String:

```
"dolphin-metadata"
```

Object:

```
{
    "type": "photo-api",
    "actions": [ "dolphin" ],
    "datatypes": [ "metadata" ]
}
```

The AS decides how this is mapped

Open question: Align request and response?

- Currently "resources" results in "access_token", should the request also be "access_token"?

Open Question: Directed Access Tokens

- We can describe “what the token’s for” but don’t have a way to say “how to use it”
- Defining usage rules in HTTP is HARD
  - Verbs, headers, parameters, URLs, etc
- Maybe a subset? Maybe an extension?

Plugging in OAuth 2

Have a place to put familiar things

```
client_id=client1
  &scope=foo%20bar
```

```
{
  "keys":
    "client1",
  "resources": [
    "foo",
    "bar"
  ]
}
```

Why not just have "client_id"?

- Identifiers should be used but not required
  - All clients identify with a key, whether registered or not
  - An identifier is a shortcut to look up the key
- OAuth 2 hangs too much on "client_id" lookup
  - Breaks ephemeral clients
  - Breaks single-user clients
  - Assumes registration
  - Confuses what a "client" even is

Why not just have “scope”?

- “Scope” is a confusing and limiting construct
  - Can’t have spaces, can’t have unicode
- RAR has to deal with how to relate to scope, resource, audience, and other parameters
- GNAP can more clearly define string-based requests as optimizations of rich requests

Making XYZ from OAuth 2

- PAR + RAR + JAR + JARM
- DPoP + PoP + MTLS +HTTPSig
- Auth Code, Device, Exchange, Refresh, Assertion, CIBA, OB/FAPI, Client Credentials, and UMA flows
- PKCE + State
- Plus a few things we haven't invented yet
- This is unwieldy at best…