XAuth

IETF 108 - GNAP WG
July 27, 2020

dick.hardt@gmail.com
Overview

• XAuth Intro

• XAuth Goals

• Key XAuth Features

• Implementation Learnings

• draft-hardt-xauth-protocol-13 (latest) vs -06 (IETF 107)

• Open Items
Parties

- User
- Resource Owner (RO)
- Client
- Grant Server (GS)
- Resource Server (RS)
Key Terms

- **Claims** - statements about the User
- **Authorizations** - Client access to a RS
- **Grant** - Collection of authorizations and/or claims issued by Grant Server (GS)
- **GS** - OAuth Authorization Server (AS) & OpenID Connect OP (OpenID Provider)
- **Interaction** - how User is directed to interact with the GS to authorize a Grant
General Sequence

Client

--(1)--- Create Grant ----------->

<--- Interaction Response ---(2)--

--(3)--- Redirect ---------------

<--- Redirect -----------------(6)-

--(7)--- Read Grant ----------->

<---------- Grant Response ---(8)--

GS

User (RO)

--(4)--> authN

<--(5)--> authZ

------->

------->

------->
XAuth GS API URI Examples

- GS URI: https://gs.example/endpoint
- Grant URI: https://gs.example/endpoint/grant/8e3a6354...
- AZ URI: https://gs.example/endpoint/authz/fad923b4...
XAuth Goals

• **Extensible** - well defined extension points

• **Migration** - from OAuth 2.0 and OpenID Connect

• **Reuse** - build on what has come before

• **Scalable** - decomposable architecture, separation of concerns

• **Simple** - simple things are simple

• **Flexible** - hard things are possible
Extensible

- User
  
  ```json
  { "user": {
    "identifiers": {...}, "claims": {...},
    "new_user_property": {...}
  },
  "client": {
    "id": "client_1", "display": {...}, "handle": "f6a60810-3d07",
    "new_client_property": {...}
  },
  "interaction": {
    "redirect": {...}, "indirect": {...}, "user_code": {...},
    "new_interaction_mode": {...}
  },
  "authorizations": {
    "token_1": { "type": "oauth", "scope": "read write" },
    "token_2": {
      "type": "new_RAR_type",
      "label": "value"
    }
  },
  "claims": {
    "oidc": {...}, "vc": {...},
    "new_claim_type": {...}
  },
  "new_objects": {...}
  }
  ```

- Client

- Interactions

- Authorizations

- Claims

- New Objects

- Client AuthN
OAuth/OIDC Migration

"client": {
    "id": <existing client id>
},
"authorizations": {
    "type": "oauth_scope",
    "scope": <existing oauth scopes>
},
"claims": {
    "oidc": {
        "userinfo": { <OIDC claims> },
        "id_token": { <OIDC claims> }
    }
}
Reuse

- JSON, TLS, HTTP
- HTTP verbs for RESTful API
- OAuth 2.0: client_id, scopes, access tokens
- OpenID Connect: client_id and claims
- OAuth Rich Authorization Request (RAR)
- JOSE for Client AuthN
Decomposed Grant Server

OPTIONS /gs

POST /gs

GET /gs/grant/...

GET /gs/authz/...

GS options

create grant

read grant

create authZ

claims

read authZ

HIGH VOLUME
Shuffle Sharding
using URI rather than message inspection

GET /gs/authz/a...
GET /gs/authz/b...
GET /gs/authz/c...
GET /gs/authz/d...
GET /gs/authz/e...
GET /gs/authz/f...
GET /gs/authz/g...
GET /gs/authz/h...

sharding across clients: a, b, c, d, e, f, g, h
Shuffle Sharding

Client d has error and takes down all its servers

---

```
GET /gs/authz/a...
GET /gs/authz/b...
GET /gs/authz/c...
GET /gs/authz/d...
GET /gs/authz/e...
GET /gs/authz/f...
GET /gs/authz/g...
GET /gs/authz/h...
```

---

Client a, b, c, e, f, g, h still have a working server
XAuth Key Features

• RESTful API
• client.id and client.handle
• interaction modes
• claims from Grant Server
• JOSE client AuthN
## RESTful API

<table>
<thead>
<tr>
<th>request</th>
<th>http verb</th>
<th>URI</th>
<th>response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Grant</td>
<td>POST</td>
<td>GS URI</td>
<td>interaction, wait, or grant</td>
</tr>
<tr>
<td>Verify Grant</td>
<td>PATCH</td>
<td>Grant URI</td>
<td>grant</td>
</tr>
<tr>
<td>Read Grant</td>
<td>GET</td>
<td>Grant URI</td>
<td>wait or grant</td>
</tr>
<tr>
<td>Read Authorization</td>
<td>GET</td>
<td>AZ URI</td>
<td>authorization</td>
</tr>
<tr>
<td>GS Options</td>
<td>OPTIONS</td>
<td>GS URI</td>
<td>metadata</td>
</tr>
</tbody>
</table>
## Advanced APIs

<table>
<thead>
<tr>
<th>request</th>
<th>http verb</th>
<th>URI</th>
<th>response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create Grant</td>
<td>POST</td>
<td>GS URI</td>
<td>interaction, wait, or grant</td>
</tr>
<tr>
<td>Verify Grant</td>
<td>PATCH</td>
<td>Grant URI</td>
<td>grant</td>
</tr>
<tr>
<td>Read Grant</td>
<td>GET</td>
<td>Grant URI</td>
<td>wait or grant</td>
</tr>
<tr>
<td>Update Grant</td>
<td>PUT</td>
<td>Grant URI</td>
<td>interaction, wait, or grant</td>
</tr>
<tr>
<td>Delete Grant</td>
<td>DELETE</td>
<td>Grant URI</td>
<td>success</td>
</tr>
<tr>
<td>Read Authorization</td>
<td>GET</td>
<td>AZ URI</td>
<td>authorization</td>
</tr>
<tr>
<td>Update Authorization</td>
<td>PUT</td>
<td>AZ URI</td>
<td>authorization</td>
</tr>
<tr>
<td>Delete Authorization</td>
<td>DELETE</td>
<td>AZ URI</td>
<td>success</td>
</tr>
<tr>
<td>GS Options</td>
<td>OPTIONS</td>
<td>GS URI</td>
<td>metadata</td>
</tr>
<tr>
<td>Grant Options</td>
<td>OPTIONS</td>
<td>Grant URI</td>
<td>metadata</td>
</tr>
<tr>
<td>Authorization Options</td>
<td>OPTIONS</td>
<td>AZ URI</td>
<td>metadata</td>
</tr>
</tbody>
</table>
Decomposed Grant Server

OPTIONS /gs

POST /gs

GET /gs/grant/...

GET /gs/authz/...

router

GS options

create grant

read grant

create authZ

claims

read authZ

HIGH VOLUME
Shuffle Sharding

using URI rather than message inspection

GET /gs/authz/a...
GET /gs/authz/b...
GET /gs/authz/c...
GET /gs/authz/d...
GET /gs/authz/e...
GET /gs/authz/f...
GET /gs/authz/g...
GET /gs/authz/h...

sharding across clients: a, b, c, d, e, f, g, h
Both `client.id` and `client.handle`

- **client.id**
  - registered client
  - identifier for any instance
  - GS has read-only access
  - thousands of ids

- **client.handle**
  - dynamic client
  - identifier for a single instance
  - GS has read/write access
  - billions of handles
Interaction Modes

• **redirect** mode
  - user redirected to GS and back to Client
  - browser or mobile app2app

• **decoupled modes**
  - **user_code** - enter code on separate device

• **indirect** - scan QR Code with phone
  
susceptible to session fixation attacks
redirect mode

Client

(A)

User

GS

(B)
indirect mode

Client

User

GS
session fixation

Client

GS

User

User
Interaction URI Examples

- **redirect** mode
  - -> completion_uri  https://client.example/complete/3902f52...
  - <- redirect_uri  https://gs.example/redirect/f52d256a...

- **decoupled modes**
  - -> information_uri  https://client.example/info
  - <- **indirect** - indirect_uri  https://gs.example/indirect/5d2f63...
  - <- **user_code** - display_uri  https://gs.example/code
Claims from Grant Server

- Reuse OpenID Connect Claims
- Reuse OpenID Connect ID Tokens
- Reuse W3C Verified Credentials
- Extend with other claim schemas
- No access token required
Claims Request

Client

(1) Request

Interaction

AS

RS

User
Claims Request

Client

User

RS

AS

(1) Request

Interaction

(2)

(3)
Claims Request

1. Request
2. Interaction
3. (3)
4. Claims
JOSE Client AuthN

- HTTP header (GET, OPTIONS, DELETE)
- HTTP body (POST, PUT, PATCH)
- include URI and HTTP method in payload
- non-repudiation of Client request
- use existing crypto libraries
- Grant Server components can verify Client independently
Implementation Confirmations

- claims.[schema] works well to determine if claims schema is supported
- authorization.type worked well to detect the authorization schema
- separate identifiers for dynamic and registered clients made client type policies easy
- interaction modes made enforcing interaction policies on request (e.g., write access required redirect mode)
- representing grants and authorizations as URIs and using methods for different APIs made routing easy
- top level objects made it easy to decompose request processing
- warnings about ignored properties looks useful, but coding it is brittle
- using embedded JWK for dynamic client was straightforward
- Grant and AZ ids in URI allowed processing before request validation
- Fairly easy to add JOSE client authentication to on top of core protocol
Implementation Learnings

• space separated scopes from OAuth 2.x is a coding thunk
• OIDC claims having a null value is a thunk. Had to use hasOwnProperty instead of just testing value
• claims.oidc.userinfo.mail is a long path to ask for a claim
• separate top level objects for authorization and authorizations is clumsy
• having a '-' in a property name is messy in JS
• must peak into JOSE object to find client info to authenticate Create Grant - had to rework design
• JSON validation is brittle. JSON Schema?
• My HTML skills suck
• My ES6 knowledge is lacking
var key = await jose.JWK.createKey('EC', 'P-256', { alg: 'ES256', use: 'sig'});

let joseBody = await jose.JWS.createSign(
    {
        format: 'compact',
        fields: {jwk: key.toJSON()} 
    },
    key
    .update(grantRequest, "utf8")
    .final();

let opt = {
    method: 'POST',
    body: joseBody,
    headers: {
        'Content-Type': 'application/jose',
        'Accept': 'application/json'
    }
}

let response = await fetch(config.gs.uri, opt);
Grant Server Routing Code

// GNAP APIs
router.options( '/', options.read);
if (config.gs?.auth?.required)
    router.use(auth.confirm);
router.post( '/', grant.create);

router.get( '/grant/:grant', grant.read);
router.put( '/grant/:grant', grant.update);
router.patch( '/grant/:grant', grant.verify);
router.delete( '/grant/:grant', grant.delete);
router.options( '/grant/:grant', grant.options);

router.get( '/az/:az', az.read);
router.put( '/az/:az', az.update);
router.delete( '/az/:az', az.delete);
router.options( '/az/:az', az.options);
Request Decomposition

```javascript
if (err = user.validate( grant )) return next(err);
if (err = interaction.validate( grant )) return next(err);
if (err = authorizations.validate( grant )) return next(err);
if (err = claims.validate( grant )) return next(err);
```
// indirect interaction not allowed if write scope requested
if (checkScope( grant.context.authorizations, 'write')) {
    grantverification = uuid();
    if (grant.context.interaction.indirect) {
        delete grant.context.interaction.indirect;
        if (0 == Object.keys(grant.context.interaction).length)
            return error.response( 403,
                'write scope not available for indirect mode')
    }
}

CLI Client QR Code

Scan and load the following QR Code:
let opt = {
  iat: utils.now(),
  nonce: uuid(),
  uri: json.uri,
  method: 'GET'
};

let joseHeader = await jose.JWS.createSign(
  {format: 'compact'}, key
).update(JSON.stringify(opt), "utf8")
  .final();

opt.headers = {
  'Accept': 'application/json',
  'Authorization': 'jose ' + joseHeader
};

let response = await fetch(json.uri, opts);
draft-hardt-xauth-protocol
-06 vs -13

• split draft into 3 documents: core, advanced, JOSE
• authorization & authorizations => authorizations
• interaction mode negotiation
• authorizations are only RAR objects ("type" being AS defined)
• client handle for dynamic clients
• editorial
Advanced Features

• Wait Response

• Update, Delete, Options Grant

• Update, Delete, Options Authorization

• GS initiated Grant Creation

• Reciprocal Delegation

• user.exists - in Grant Request

• interaction.keep - multi-step interaction
Open Items

- SecEvent Subject Identifiers in user object?
- Add CIBA like interaction mode?
- array of RAR objects per authorization request?
- example OIDC userinfo RAR
  
  "authorizations":{
    "type": "<name_space>/oidc_userinfo",
    "claims": { OIDC claims }
  }

- Simplify OIDC claims
  
  "claims":{
    "oidc_userinfo" : { OIDC claims },
    "oidc_id_token" : { OIDC claims }
  }
Questions?