Managing authorization grants beyond OAuth 2

Fabien Imbault¹, Justin Richer² and Aaron Parecki³

Abstract: The Grant Negotiation and Authorization Protocol, also known as GNAP, is currently being formulated in an IETF working group. GNAP gives the opportunity to reflect on the strengths and weaknesses of OAuth 2, and highlights the new directions to improve digital access. We compare with the approach taken by OAuth 2 and show that designing authorization servers primarily as “token issuers” provides insightful consequences for security and privacy.

Keywords: authorization protocol, OAuth 2, GNAP

1 Lessons from OAuth2

1.1 A short history

The year was 2012, and an authorization protocol called OAuth 2 (Open Authorization 2) swept the web, allowing users to use security providers to easily log in to websites. Coupled with OpenID, OAuth 2 enables an end-user to “authenticate with” one of its providers (google, facebook, github, etc.) to a completely different website or application, therefore reducing the need to define yet another password. OAuth 2 aims to solve the delegated authorization problem. Delegation happens when a third party application, acting on behalf of a natural person, requests access to a protected resource. The naive way to solve this problem is for the natural person to give its password to the third party, but sharing passwords is a security risk and must be avoided. OAuth 2 defines flows to grant access without having to share secrets.

Solving the delegation use case had such an impact because OAuth 2 landed at a time where Application Programming Interfaces (API) really became mainstream. In 2020, 83% of the internet traffic was due to APIs (compared to the remaining 17% through HTML). Cloud based companies in particular found it convenient to better secure the access to their protected API endpoints. As Gartner points out, this trend is accelerating [Ze19]: “90% of web-enabled applications have more surface area for attack in the form of exposed APIs rather than the UI, up from 40% in 2019.” Relying on a common framework enabled easier integrations across services too, as exemplified by the

¹ fabien.imbault@gmail.com
² jricher@mit.edu
³ aaron@parecki.com
widespread use of zapier amongst software as a service providers. It also reduces the risk of vendors inventing their own security mechanism. Most major services now support OAuth 2, often associated with OpenID Connect for single-sign on. OAuth 2 is heavily used to protect API first services, such as open banking. The decade experience that exists with OAuth 2 and its extensions got recently consolidated into an OAuth 2.1 draft version. If approved, this update will obsolete certain parts of OAuth 2.0 and mandate security best practices [Fe16][Pa19].

1.2 OAuth 2 Pillars

The “core” OAuth 2.0 spec, RFC 6749 [Ha12], isn’t a specification, it’s a “framework” you can use to build specifications from. It defines roles and a base level of functionality, but leaves a lot of implementation details unspecified or optional. The IETF OAuth Working Group has published many additional specifications to fill in the missing pieces. Implementers need to decide which grant types to support, whether or not refresh tokens are one-time use, and even whether access tokens should be bearer tokens or use some sort of signed token mechanism.

The protocol seeks to enable a separation of concerns, decoupling authentication from authorization. This is a significant difference to previous protocols, such as Kerberos, Radius or SAML. OpenID Connect (OIDC) is the de-facto identity layer on top of OAuth 2.0, with extensions emerging from the decentralized identity space.

A complete introduction to the capabilities of OAuth2 is beyond the objective of this article, the interested reader may refer to the book “OAuth2 in Action”[Ri17].

1.3 An example of limitation

Despite its widespread use and impressive success, OAuth 2 also has downsides. A complete discussion of the current limitations is beyond the scope of this short article, but the following example illustrates the fact that human centric authorizations cannot be implemented through OAuth 2 or its UMA2 variant [Um18]:

a) A credentialed doctor (Dr. Bob) uses a secure wallet (capable of a non-repudiable signature) to make a request (relying party credentials, scope of resource server access, purpose of access) to patient Alice's authorization server;
b) The AS responds with a scoped capability and holds Bob accountable for its invocation;
c) Dr. Bob passes that capability to his employer institution or to another healthcare partner. Dr. Bob may attenuate the capability before or after it is passed to the system;
d) Another physician in the team, Dr. Carol, signs-in to the employer system and clicks on the capability associated with Alice;
e) The client (e.g. a mobile application) used by the healthcare team (Dr. Bob or Dr. Carol) presents the capability to the protected information and gains scoped accesses to
the resource. Organizational policies would likely require an audit trail that includes the doctors’ credentials and/or the root of trust of a software statement presented by the client to ensure its authenticity.

OAuth 2 would have trouble handling such a complex but realistic and common scenario, possibly requiring coordination between several ASs and resource servers, and involving delegations and policies between multiple users (the doctors) distinct from the resource owner (the patient). By focusing on the usability of privacy and security protocols within the real-world contexts in which they have to operate, we target a human-centric design [Sa05]. We therefore propose to take a fresh look at how to design a delegation protocol.

2 Alternative design principles with GNAP

In this section, we explain how a new protocol currently being specified within the IETF GNAP (Grant Negotiation and Authorization Protocol) working group, goes away from some of the current OAuth 2 assumptions and limitations. The core specification document is publicly available as a draft [Ie20]. Many changes are still expected before the specification is officially published, but the general design principles have received consensus from the project charter. A formal terminology [Ie21] has been approved. Early versions of the draft are already implemented as open source projects by different stakeholders, to ensure those concepts are practically sound (as per the unofficial IETF motto: “we believe in rough consensus and running code”). Non-goals have been made explicit. OAuth 2 comes with known benefits and GNAP also doesn’t intend to replace OAuth 2 or its extensions. An appendix in the GNAP core specification defines how to retrofit scopes and client_id from existing OAuth 2 systems and enable a progressive roll-out.

2.1 Cryptography based security

OAuth 2 uses shared bearer secrets, including the client_secret and access token, and advanced authentication and sender-constraining have been built on after the fact in inconsistent ways. In GNAP, all communication between the client instance and AS is bound to a key held by the client instance.

GNAP uses the same cryptographic mechanisms for both authenticating the client (to the AS) and binding the access token (to the resource server and the AS). It allows extensions to define new cryptographic protection mechanisms, as new methods are expected to become available over time. GNAP does not have a notion of “public clients” because key information can always be sent and used dynamically in addition to being pre-registered.
2.2 Interaction flexibility

OAuth 2 generally assumes the user has access to a web browser. The type of interaction available is fixed by the grant type, and the most common interactive grant types start in the browser.

GNAP allows a client instance to list different ways that it can start and finish an interaction, and these can be mixed together as needed for different use cases. GNAP interactions can use a browser, but don’t have to. Methods can use inter-application messaging protocols, out-of-band data transfer, or anything else. GNAP also allows extensions to define new ways to start and finish an interaction, as new methods and platforms are expected to become available over time. GNAP is designed to allow these users to be two different people, but still works in the optimized case of them being the same party.

2.3 Intent registration and inline negotiation

OAuth 2 uses different “grant types” that start at different endpoints for different purposes. Many of these require discovery of several interrelated parameters. GNAP requests all start with the same type of request to the same endpoint at the AS. Next steps are negotiated between the client instance and AS based on software capabilities, policies surrounding requested access, and the overall context of the ongoing request.

GNAP defines a continuation API that allows the client instance and AS to request and send additional information from each other over multiple steps. This continuation API uses the same access token protection that other GNAP-protected APIs use.

2.4 Client instances

OAuth 2 requires all clients to be registered at the AS and to use a client_id known to the AS as part of the protocol. This client_id is generally assumed to be assigned by a trusted authority during a registration process, and OAuth 2 places a lot of trust on the client_id as a result and requires it throughout the protocol. Dynamic registration allows different classes of clients to get a client_id at runtime, even if they only ever use it for one request.

Instead of a client_id (related to a pre-registered client software), GNAP relies on client instances (identified by their key). GNAP allows the client instance to present an unknown key to the AS and use that key to protect the ongoing request. It also allows to define attestation mechanisms for the client software (for instance, the organization the client represents, a specific version, the posture of the device the client is installed on, etc.). GNAP’s client instance identifier mechanism allows for pre-registered clients and dynamically registered clients to exist as an optimized case without requiring the identifier as part of the protocol at all times.
2.5 Expanded delegation

OAuth 2 defines the “scope” parameter for controlling access to APIs. This parameter has been co-opted to mean a number of different things in different protocols, including flags for turning special behavior on and off, including the return of data apart from the access token. The “resource” parameter and RAR extensions expand on the “scope” concept in similar but different ways [Lo20].

GNAP defines a rich structure for requesting access and supports string references as an optimization. GNAP defines methods for requesting directly-returned user information, separate from API access. This information includes identifiers for the current user and structured assertions.

2.6 Privacy by design

OAuth 2 has no protection against a curious AS.

GNAP intends to provide privacy preserving mechanisms based on data minimization and untraceability. Those mechanisms could either target a single AS or multiple ASs in order to reduce centralization and improve scalability.

3 Conclusion

This article provides a comparison of OAuth 2 and the more recent GNAP authorization protocol. The later covers simple delegation in a more consistent way but also enables advanced cases between various stakeholders involved in sensitive application domains.

In OAuth2, it is assumed that the AS is the device that’s authenticating the user, collecting consent, managing the client’s registration, and creating an access token based on whatever set of rights that are associated with all of those things. In UMA2, this is turned around by letting the resource owner present a bunch of “claims” interactively, but still at the AS. With both of these, a key aspect remains: the AS needs to gather necessary information, and issue the access token (or identifiers/assertions). Because of how GNAP works, the client software has a better opportunity to present information to the AS, either directly in the request, through external parties or by introducing the AS to another software component during the “interaction” phase. So GNAP asks, what if we think of authorization server(s) primarily as a “token issuer(s)”?

Beyond GNAP, reflecting on the current assumptions and limitations of OAuth 2 is a worthwhile exercise that would require a closer partnership between practitioners and academia. In particular, a better understanding of the security and privacy guarantees would benefit the general public and the regulatory bodies.
Disclaimer

The authors are co-editors of the IETF GNAP specification and would like to thank the participants of the working group. Please note that this paper has been written independently and has not been endorsed by the IETF.

This work derives partially from a project (median) which has received funding from the European Union’s Horizon 2020 research and innovation programme under the NGI_TRUST grant agreement no 825618.

Bibliography


[Sa05] Sasse, A.; Flechais, I.; Usable Security. Why Do We Need It? How Do We Get It? In: Cranor, LF and Garfinkel, S, (eds.) Security and Usability: Designing secure systems that people can use, O'Reilly, 2005
