OAuth is DAC. What do you do for MAC?

Johan Peeters
Independent

Abstract

Such is the frustration of the development community with SAML, that most new projects turn to OAuth. Yet the goals of the OAuth are completely different to SAML's: the former gives the end user control over who has access to their resources, while the latter is mainly used to enforce compliance to security policy. Most projects need both, so vendors are building ad-hoc extensions to their OAuth authorization servers to meet the need for mandatory access control and developers are piecing together elements from the OAuth and OIDC specs, proprietary extensions and custom code.
This paper is a call for guidance on how to integrate discretionary access control as provided by OAuth and mandatory access control as demanded by business. It explores some common practices which could be adopted as standards and, conversely, how current standards could be leveraged.

Position

There is a need for a widely supported access control protocol for Mandatory Access Control (MAC) that integrates well with OAuth.

DAC versus MAC

Access control models are often partitioned into Mandatory Access Control (MAC) and Discretionary Access Control (DAC). The distinguishing characteristic of the former is the enforcement of centrally administered security policies. In other words, whether an entity can or cannot perform an action on a resource is governed by rules imposed by central authority. In DAC models, on the other hand, this decision is taken by the resource owner.
MAC is usually associated with high-assurance deployments such as the military and intelligence communities. However, MAC also fits the aspirations of most corporations and is widely used in the guise of Role-Based Access Control (RBAC). In RBAC, subjects are assigned roles which, are in turn assigned permissions to perform actions on resources. Usually, an administrator makes these assignments at the behest of a central authority. Thus common corporate practice protects the crown jewels with MAC, while DAC may be used by employees protecting work in progress.
Social media and cloud services have blurred the right to control data. As users entrust their data to corporations such as Facebook, Google, Microsoft and LinkedIn, they want DAC for their data. OAuth fills that gap. But even in this brave new world of user-controlled data the need for MAC remains. Consider the example of customers of an e-commerce site who can

selectively expose account details to social media such as purchases or wishlist. While DAC, and hence OAuth, is well-suited for this use case, a helpdesk probably also needs access to customers' accounts. Which helpdesk worker can do what to which account is a matter for MAC. Donald Trump's temporary ban from Twitter is an interesting illustration of how defective access control for helpdesk workers can have far-reaching consequences.

OpenID Connect

OpenID Connect (OIDC) has superseded OAuth as most, if not all, recent authorization servers support OIDC. Since OIDC is built on top of OAuth 2.0, support for OIDC includes support for OAuth. Conversely, choosing OAuth for access control, also includes OIDC, regardless of whether identity services are required. So, even though OIDC is not an authZ protocol, it is effectively being used as such in many cases. This practice is reflected and reinforced by referring to an OIDC Security Token Service as an 'authorization server'. Ironically, the 'OAuth is not for authN' meme has resulted in overwhelming use of OIDC for authZ.

Scope

The intention of the scope parameter is to communicate the scope of the access requested by the client to the authorization server. This allows the authorization server to enter into a dialog with end users to verify that they are indeed prepared to grant requested permissions. OIDC standardizes the following scope tokens: `openid`, `profile`, `email`, `address` and `phone`. It is common for an OIDC client to specify the `openid` scope alongside access token scopes, effectively requesting both an ID and access token. There is no standardization for access token scopes, which leads to projects developing proprietary scope syntax and semantics. For example, it is frequently suggested to use scope tokens of the `<action>:<resource>` form, but how this affects issued tokens or how this leads to an access control decision remains unclear.
The authorization server may ignore the scope string - RFC 6749 explicitly mentions that this may happen 'based on the authorization server policy or the resource owner's instructions'. While the latter is aligned with the goal of user-administered access control, the former appears to be making allowances for a MAC model. It is of course entirely logical that the authorization server should not only answer the question 'does the user agree that the requesting client is giving these access permissions?', but also 'is the access requested by the user in line with security policy?'

JWT

In contrast to OAuth, OIDC standardizes the security token format. OIDC's ID token is a JWT, and, in the absence of a prescribed format for the OAuth access token, JWT has become the de-facto standard for the access token.

It would be helpful if standards would specify JWT as a possible, or preferred, format for access tokens. Like OIDC, OAuth standards could then distinguish between mandatory, optional and custom claims.
Standards which specify optional claims with well-defined semantics can be very helpful in establishing a blueprint, or checklist, of common concerns in the authN/Z space. The `azp` claim, discussed in the next section, is an example of a claim that serves as a welcome reminder of an easily overlooked aspect.

Audience and authorized party

OIDC stipulates that the `aud` claim must be present in an ID token. Moreover, a client or relying party must validate that its client ID is amongst the `aud` values. Such requirement makes perfect sense from an authZ perspective but is absurd if the intended use is authN: permissions may vary according to the target resource server, user identity does not.
This is not a complaint about the mandatory presence and validation of `aud` in an OIDC token, but more of an observation that the 'authN, not authZ' narrative does not quite fit the spec. On the contrary, `aud` is frequently embraced to support communication of a security policy. For example, in a project that makes pre-configured computing infrastructure available to small businesses, the aud identifies the intended target infrastructure - the authorization server only issues tokens for resources that the user had access to.
Similarly, the optional `azp` claim has been standardized by OIDC. This claim '`is only needed when the ID Token … is different than the authorized party`.' In other words, a token is expected to contain this claim when a client forwards it to a resource server. Again, this claim seems to make more sense in an authZ than an authN scenario.

RBAC

The MAC gap in the OIDC authorization server spec has been filled by vendors in an ad-hoc fashion. Unsurprisingly, given the industry's familiarity with RBAC, roles feature heavily. Building on this momentum, a straightforward way of supporting MAC with OAuth access tokens is to specify a standard 'roles' claim. It could be argued that, given the limitations of RBAC, a more powerful access control model is desirable. Many have argued for ABAC. However, support for RBAC and ABAC are not mutually exclusive and both could be pursued in parallel.

Conclusion

OAuth may broadly have achieved its aims to provide users with access control over their resources, but there are some loose ends that need tidying. Adoption of JWT as the format for access tokens is a no-brainer given current practice. This would also enable inclusion of claims supporting MAC. Strong candidates would be some of the claims of the OIDC ID token such as `aud` and `azp`.

IAM projects seem to be reinventing the wheel over and over again. Better guidance, whether in the form of standards documents or otherwise, could call a stop to this waste. Areas where such guidance might be particularly fruitful include the syntax of access token scopes, how requested scopes impact of an access token and its effect on the authorization decision.
RBAC is by no means universally acclaimed. However, given its dominance in industry and wide support among authorization servers, standardization of JWT claims to support RBAC would be a boon for interoperability. If claims supporting more expressive, or simpler, access control models, such as ABAC, can be standardized as well, so much the better. However, given the need for a shared vision on the nature and representation of common attributes, this is likely to be a much more involved exercise.

References

OpenID Connect Core 1.0, https://openid.net/specs/openid-connect-core-1_0.html

JSON Web Token (JWT), https://tools.ietf.org/html/rfc7519

The OAuth 2.0 Authorization Framework, https://tools.ietf.org/html/rfc6749