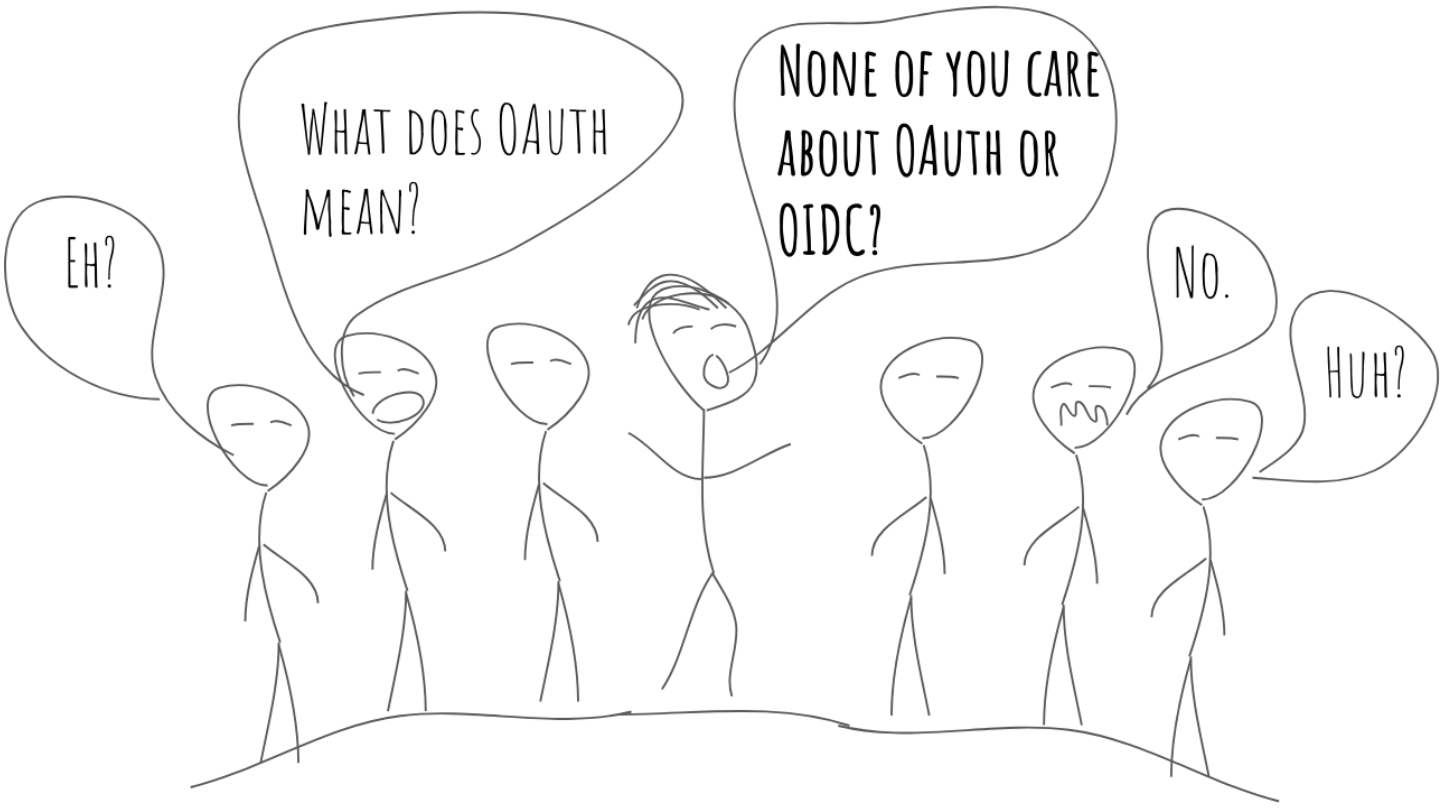
January 23, 2019

Nobody Cares About OAuth or OpenID Connect

Randall Degges



If you're reading this post, there's a good chance that you're a web developer who's very interested in web security. You've probably heard about OAuth or OpenID Connect (OIDC) before. You may have even used them at some point in your career.

But here's the thing: *almost* nobody actually cares about OAuth or OIDC. Not you, not me, and not even other developers in the security industry.

To understand why nobody cares about these two critical web standards, you need to understand some history.

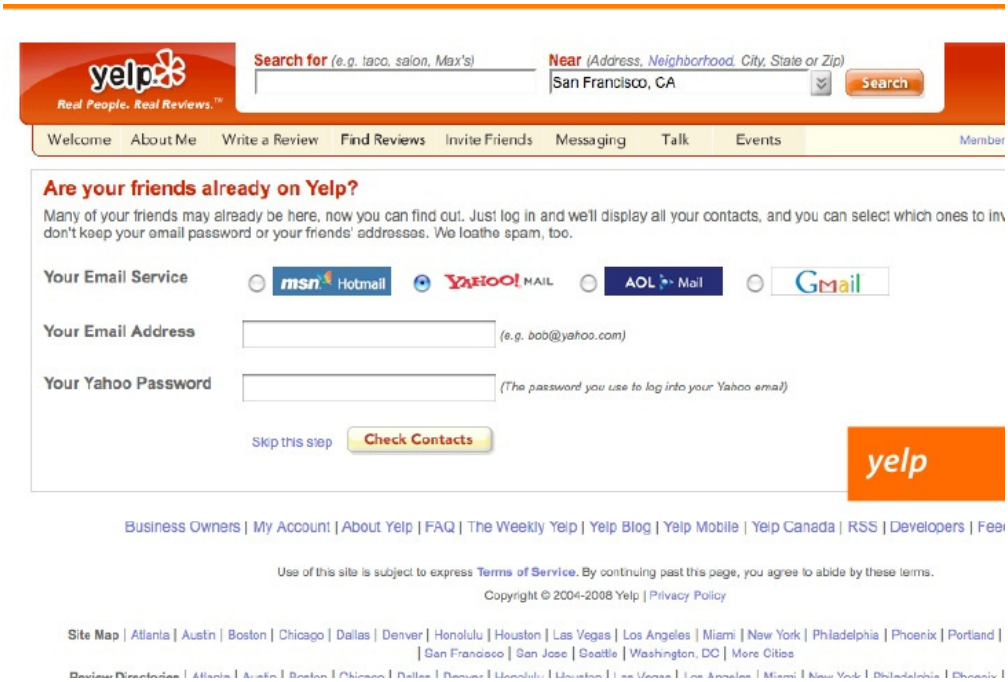
Why OAuth Exists

Back in pre-2007, there was no way for developers to build apps that needed to securely access user data in another service.

Take Yelp, for example. Back in the day, if you wanted to find your friends on Yelp, you'd have to:

- Tell Yelp what email provider you use
- Give Yelp your email address
- Give Yelp your email password

In short: Yelp would need you to give them all of your sensitive credentials so they could then start making requests to your email provider directly, download your contacts, then do something with that data.

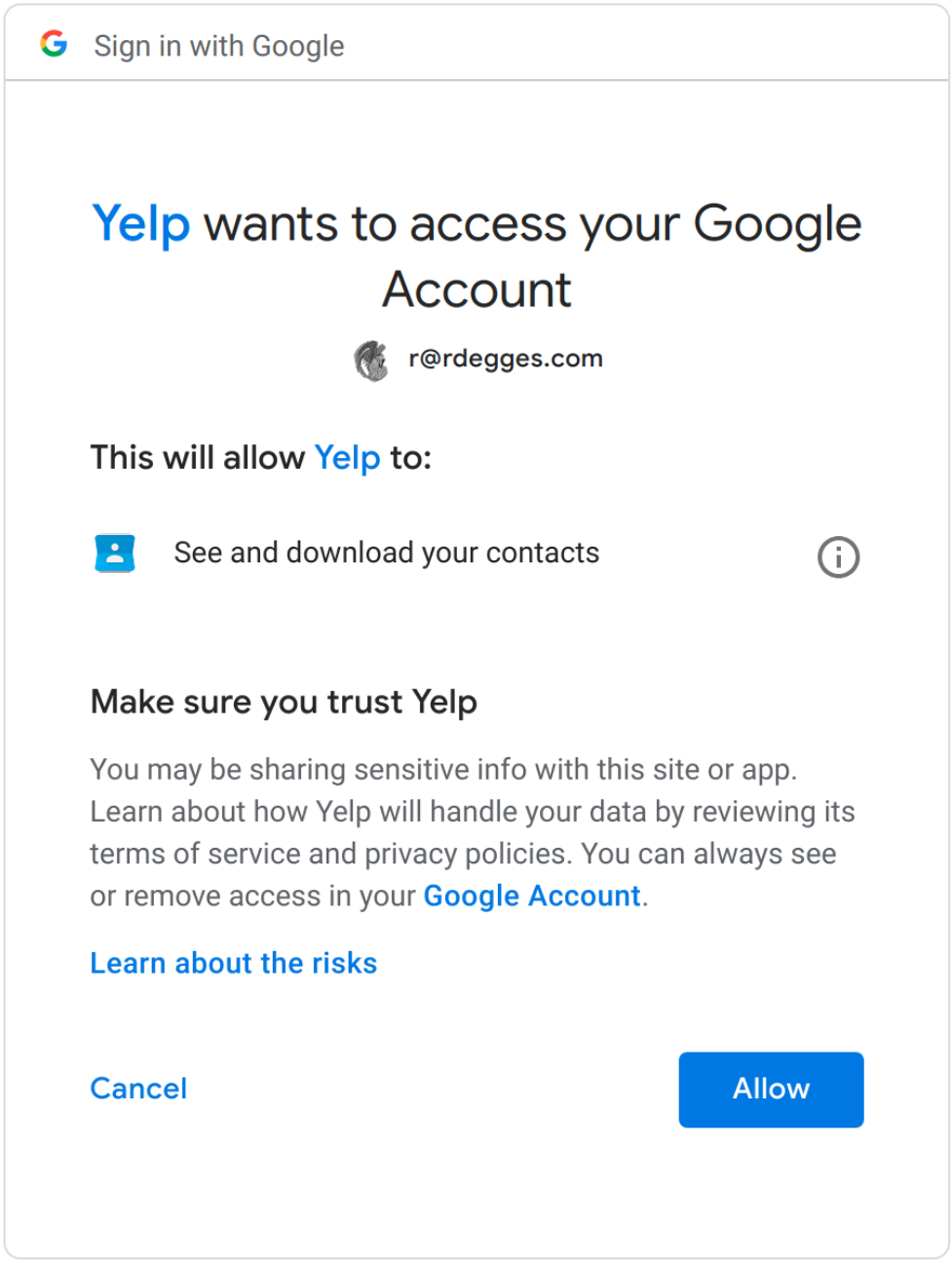


If it isn't immediately obvious: *this is not good*. You (as a user) shouldn't give your sensitive login credentials out to other companies; it puts you at risk. The fewer people that have access to your login information, the better.

Leading up to 2007, some developers at Twitter were trying to solve this problem and basically said *"You know what... There isn't really a great way to solve this whole delegated authorization problem right now. Maybe we can do something about that."* And just like that, the OAuth discussion group was formed.

The OAuth group eventually released a spec, and that spec eventually changed the way authorization works on the web.

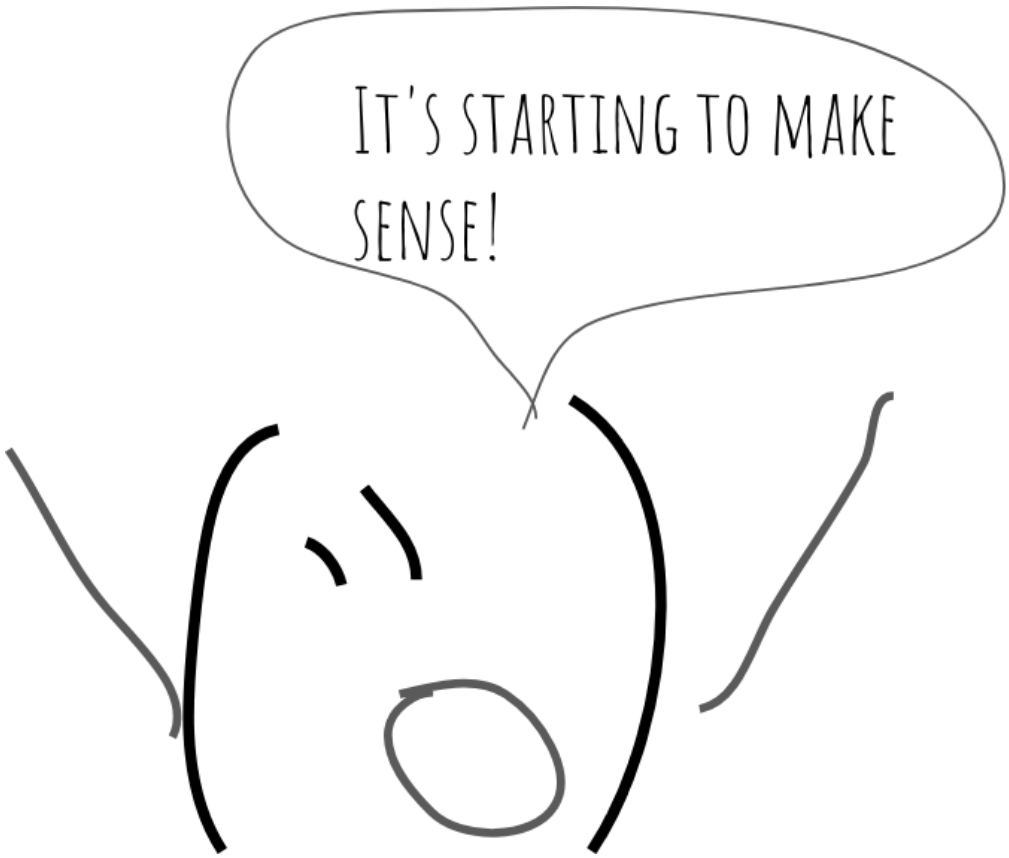
Nowadays, that same Yelp workflow has been replaced with a much safer flow (pictured below).



Not only is this new flow a lot safer–it's also a lot simpler. When Yelp wants to access your Gmail contacts, they simply redirect you to Google (which you are presumably already logged into), where you're then greeted by the pictured consent screen, telling you what specific data Yelp will be able to access in your account if you allow them to.

Yelp doesn't need to worry about storing your sensitive data and you don't need to worry about Yelp misusing your credentials: it's a win-win for everyone. In this way, OAuth made the web a much safer place.

Why OpenID Connect Exists



OAuth solved a lot of issues, but it didn't solve everything.

Authorization is all about finding out who is allowed to access certain data and functionality. *Authentication* is all about logging in. Unfortunately, these are two completely different things, and this is where the trouble started.

While OAuth solved the authorization problems present on the web at the time, it didn't even attempt to tackle authentication issues. OAuth's lack of authentication guidance led to a number of confusing, complex integration scenarios, which is precisely why OpenID Connect (OIDC) was created.

OIDC is a newer standard that extends OAuth, adding support for authentication. It doesn't change any of the OAuth behaviors, all it does is add some extra stuff to solve both authentication *and* authorization challenges that most web developers face.

The OIDC specification also clarified a number of ambiguities in the original OAuth spec, giving implementors a much clearer understanding of what technologies to use under the hood, how to validate requests and a number of other things that aren't worth diving into in this article.

Why Things Went Wrong with OAuth and OpenID Connect

Things are sounding pretty good so far, right? You might be wondering, *"What's the problem?"*

After all, there were two main problems on the web, authorization and authentication, and they were both solved by OAuth and OIDC. End of story.

But not so fast... Like most things, the devil is in the details.

Both OAuth and OIDC are fundamentally complicated: they solve complex web security problems in a number of different environments.

The OAuth and OIDC specs (and extensions) cover authentication and authorization for:

- Users logging into a server-side web application
- Users logging into a client-side web application
- Users logging into a native mobile application
- Users logging into a TV/device application
- Server-to-server API authorization

- Etc.

These use case scenarios are translated into a concept called *grant types* in the OAuth specification, and each one works differently and has different security profiles that the implementor needs to be aware of.

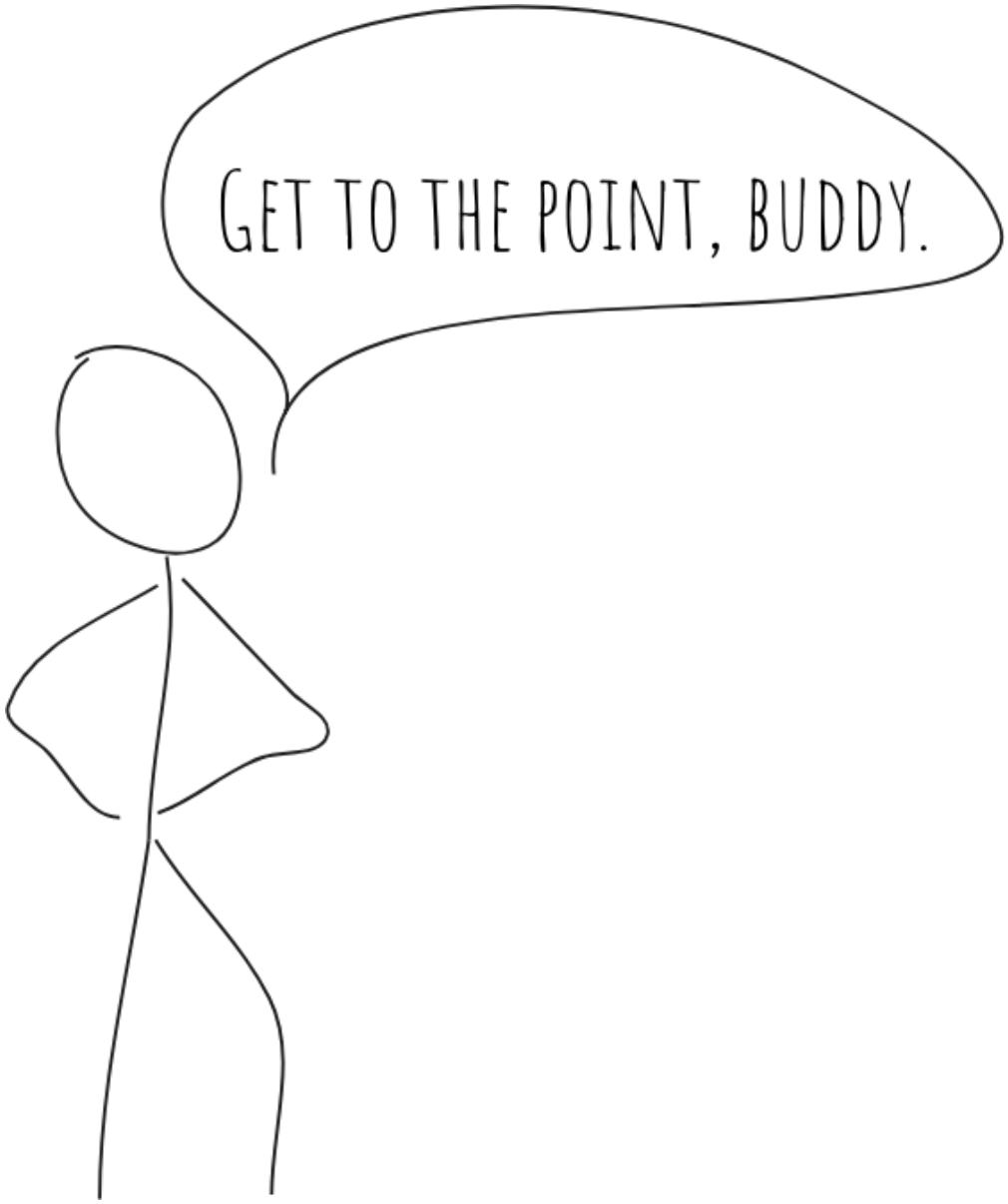
In addition to grant types, implementors must also understand tokens (specifically, JSON Web Tokens) (which are a part of the OIDC specification), which adds additional learning curves to the OAuth/OIDC implementor.

In order to successfully implement OAuth/OIDC in your environment, you (as a developer) need to fully understand:

- What the different OAuth grant types are
- Which grant types you should be using in your application in which scenarios
- What JWTs are
- How to generate JWTs securely and keep your keys safe
- How to sign (and optionally encrypt) your JWTs: what options do you use?
- How to validate JWTs
- How to assess the security repercussions of using JWTs in different ways
- And lots more

As you can see above, the learning curve can be a lot, even for experienced developers. And this is exactly the problem.

Why Nobody Cares About OAuth and OpenID Connect



OAuth and OIDC are complicated, and it takes a lot of time and effort to understand and use them properly without opening yourself up to exploitation.

The reason nobody cares about OAuth and OIDC is that OAuth and OIDC aren't what developers are interested in. The only thing developers are *actually* interested in is what OAuth and OIDC help with, **authentication and authorization**.

99.99% of developers out there don't know (or want to know) anything about OAuth, OIDC, or any other security specifications. All they want to do is find the simplest and most straightforward way to support user authentication and authorization in their application. They don't care about standards, specifications, grant types, JWTs, or scopes and timeouts – all they want to do is log a user in and check to see what permissions they have.

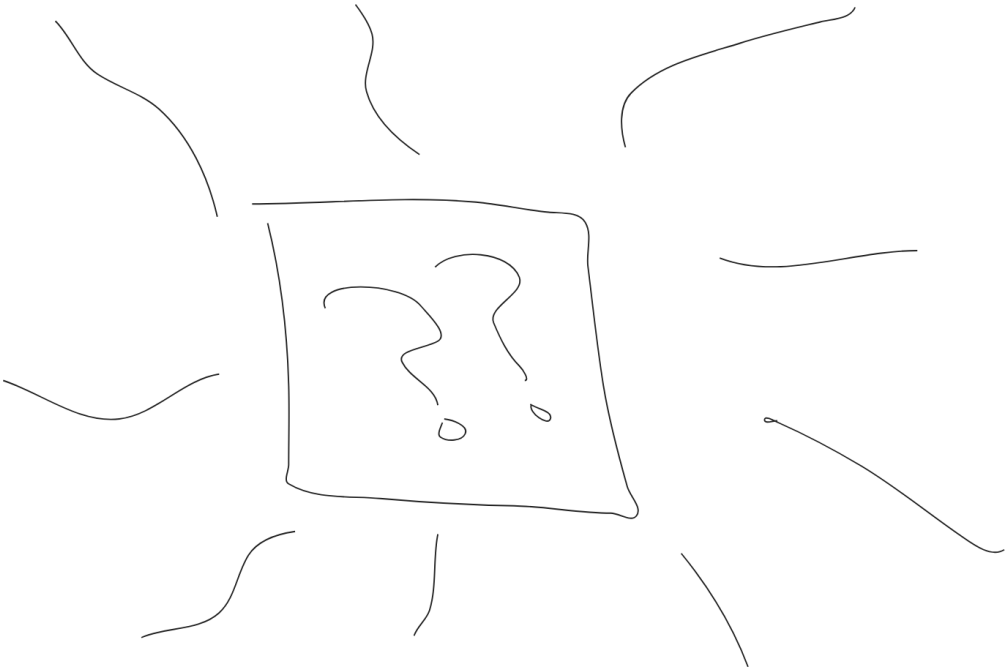
What's funny is that in the cryptography world, there's a popular adage that is often repeated:

> "Don't roll your own crypto."

This adage has been repeated for as long as I can remember, and basically means: let the experts handle the hard stuff (cryptography) because it's incredibly hard to get right, and there are a million and a half ways to mess it up.

In the web development world, however, people do the equivalent of rolling their own crypto every day! And often times, not by choice! Trying to figure out and understand what types of OAuth/OIDC to use for your project, how to mint and manage tokens properly, and how to plug everything together with claims, scopes, token timeouts, etc., can all be very complex and time-consuming.

What Developers Actually Care About



It's our job in the security community to keep developers safe and make their lives easier. And I'd argue that as of right now we haven't done a good enough job of this.

Instead of building new OAuth and OIDC libraries, why not build new authentication and authorization libraries? Why not build tooling that abstracts away the concepts of Client IDs and Client Secrets, PKCE, grant types and flows, and JSON Web Tokens?

With the state of tooling right now, web developers are essentially *forced* to learn about OAuth and OIDC and are burdened with the need to understand how these standards work and how to (hopefully) apply them properly to their application. It isn't a great system.

This is one of the reasons why, here at Okta, even though our entire platform is built on top of OAuth and OIDC, we spend tons of time and effort trying to build abstractions (in the form of client libraries) to

hide those complexities and make securing your web applications simpler.

While OAuth and OIDC are certainly useful and important, the reality of the situation today is that almost nobody cares about OAuth and OIDC. Developers don't want more OAuth and OIDC libraries and documentation in their lives: they want less of it.

What about you? Do you care about OAuth and OIDC? Or do you just want to focus on authentication and authorization and not bother getting involved with standards? What do you see as the future of web authentication and authorization? Drop us a tweet and let us know, or leave a comment below.