Zeropasswords

What's wrong with OAuth 2.0?

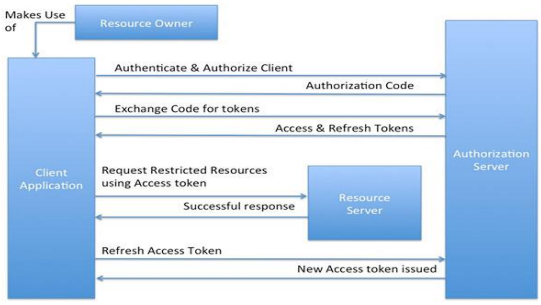
by

Eran Hammer

Nikos Leoutsarakos
wrote the conclusion

OAuth 2.0 and the Road to Hell

They say the road to hell is paved with good intentions. Well, that's OAuth 2.0. Last month I reached the painful conclusion that I can no longer be associated with the OAuth 2.0 standard. I resigned my role as lead author and editor, withdraw my name from the specification, and left the working group. Removing my name from a document I have painstakingly labored over for three years and over two dozen drafts was not easy. Deciding to move on from an effort I have led for over five years was agonizing.



There wasn't a single problem or incident I can point to in order to explain such an extreme move. This is a case of death by a thousand cuts, and as the work was winding down, I've found myself reflecting more and more on what we actually accomplished. At the end, I reached the conclusion that OAuth 2.0 is a bad protocol. WS-* bad. It is bad enough that I no longer want to be associated with it. It is the biggest professional disappointment of my career.

All the hard fought compromises on the mailing list, in meetings, in special design committees, and in back channels resulted in a specification that fails to deliver its two main goals – security and interoperability. In fact, one of the compromises was to rename it from a protocol to a framework, and another to add a disclaimer that warns that the specification is unlike to produce interoperable implementations !!

When compared with OAuth 1.0, the 2.0 specification is more complex, less interoperable, less useful, more incomplete, and most importantly, less secure.

To be clear, OAuth 2.0 at the hands of a developer with deep understanding of web security will likely result in a secure implementation. However, at the hands of most developers – as has been the experience from the past two years – 2.0 is likely to produce insecure implementations.

How did we get here?

At the core of the problem is the strong and unbridgeable conflict between the web and the enterprise worlds. The OAuth working group at the IETF started with strong web presence. But as the work dragged on (and on) past its first year, those web folks left, along with every member of the original 1.0 community. The group that was left was largely all enterprise… and me.

The web community was looking for a protocol very much in-line with 1.0, with small improvement in areas that proved lacking: simplifying signature, adding a light identity layer, addressing native applications, adding more flows to accommodate new client types, and improving security. The enterprise community was looking for a framework they can use with minimal changes to their existing systems, and for some, a new source of revenues through customization. To understand the depth of the divide – in an early meeting the web folks wanted a flow optimized for in-browser clients while the enterprise folks wanted a flow using SAML assertions.

The resulting specification is a designed-by-committee patchwork of compromises that serves mostly the enterprise. To be accurate, it doesn't actually give the enterprise all of what they asked for directly, but it does provide for practically unlimited extensibility. It is this extensibility and required flexibility that destroyed the protocol. With very little effort, pretty much anything can be called OAuth 2.0 compliant.

Under the Hood

To understand the issues in 2.0, you need to understand the core architectural changes from 1.0:

Unbounded tokens - In 1.0, the client has to present two sets of credentials on each protected resource request, the token credentials and the client credentials. In 2.0, the client credentials are no longer used. This means that tokens are no longer bound to any particular client type or instance. This has introduced limits on the usefulness of access tokens as a form of authentication and increased the likelihood of security issues.

Bearer tokens - 2.0 got rid of all signatures and cryptography at the protocol level. Instead it relies solely on TLS. This means that 2.0 tokens are inherently less secure as specified. Any improvement in token security requires additional specifications and as the current proposals demonstrate, the group is solely focused on enterprise use cases.

Expiring tokens - 2.0 tokens can expire and must be refreshed. This is the most significant change for client developers from 1.0 as they now need to implement token state management. The reason for token expiration is to accommodate self-encoded tokens – encrypted tokens which can be authenticated by the server without a database look-up. Because such tokens are self-encoded, they cannot be revoked and therefore must be short-lived to reduce their exposure. Whatever is gained from the removal of the signature is lost twice in the introduction of the token state management requirement.

Grant types - In 2.0, authorization grants are exchanged for access tokens. Grant is an abstract concept representing the end-user approval. It can be a code received after the user clicks 'Approve' on an access request, or the user's actual username and password.

The original idea behind grants was to enable multiple flows. 1.0 provides a single flow which aims to accommodate multiple client types. 2.0 adds significant amount of specialization for different client type.

Indecision Making

These changes are all manageable if put together in a well-defined protocol. But as has been the nature of this working group, no issue is too small to get stuck on or leave open for each implementation to decide. Here is a very short sample of the working group's inability to agree:

- No required token type
- No agreement on the goals of an HMAC-enabled token type
- No requirement to implement token expiration
- No guidance on token string size, or any value for that matter
- No strict requirement for registration
- Loose client type definition
- Lack of clear client security properties
- No required grant types
- No guidance on the suitability or applicability of grant types
- No useful support for native applications (but lots of lip service)
- No required client authentication method
- No limits on extensions

On the other hand, 2.0 defines 4 new registries for extensions, along with additional extension points via URIs. The result is a flood of proposed extensions. But the real issue is that the working group could not define the real security properties of the protocol. This is clearly reflected in the security consideration section which is largely an exercise of hand waving. It is barely useful to security experts as a bullet point of things to pay attention to.

In fact, the working group has also produced a 70 pages document describing the2.0 threat model which does attempt to provide additional information but suffers from the same fundamental problem: there isn't an actual protocol to analyze.

Reality

In the real world, Facebook is still running on draft 12 from a year and a half ago, with absolutely no reason to update their implementation. After all, an updated 2.0 client written to work with Facebook's implementation is unlikely to be useful with any other provider and vice-versa. OAuth 2.0 offers little to none code re-usability.

What 2.0 offers is a blueprint for an authorization protocol. As defined, it is largely useless and must be profiles into a working solution – and that is the enterprise way. The WS-* way. 2.0 provides a whole new frontier to sell consulting services and integration solutions.

The web does not need yet another security framework. It needs simple, well-defined, and narrowly suited protocols that will lead to improved security and increased interoperability. OAuth 2.0 fails to accomplish anything meaningful over the protocol it seeks to replace.

To Upgrade or Not to Upgrade

Over the past few months, many asked me if they should upgrade to 2.0 or which version of the protocol I recommend they implement. I don't have a simple answer. If you are currently using 1.0 successfully, ignore 2.0. It offers no real value over 1.0 (I'm guessing your client developers have already figured out 1.0 signatures by now).

If you are new to this space, and consider yourself a security expert, use 2.0 after careful examination of its features. If you are not an expert, either use 1.0 or copy the 2.0 implementation of a provider you trust to get it right (Facebook's API documents are a good place to start). 2.0 is better for large scale, but if you are running a major operation, you probably have some security experts on site to figure it all out for you.

Now What?

I'm hoping someone will take 2.0 and produce a 10 page profile that's useful for the vast majority of web providers, ignoring the enterprise. A 2.1 that's really 1.5. But that's not going to happen at the IETF. That community is all about enterprise use cases and if you look at their other efforts like OpenID Connect (which too was a super simple proposal that turned into almost a dozen complex specifications), they are not capable of simple.

I think the OAuth brand is in decline. This framework will live for a while, and given the lack of alternatives, it will gain widespread adoption. But we are also likely to see major security failures in the next couple of years and the slow but steady devaluation of the brand. It will be another hated protocol you are stuck with.

At the same time, I am expecting multiple new communities to come up with something else that is more in the spirit of 1.0 than 2.0, and where one use case is covered extremely well. OAuth 1.0 was all about small web startups looking to solve a well-defined problem they needed to solve fast. I honestly don't know what use cases OAuth 2.0 is trying to solve any more.

Final Note

This is a sad conclusion to a once promising community. OAuth was the foster child of small, quick, and useful standards, produced outside standards bodies without all the process and legal overhead.

Our standards making process is broken beyond repair. This outcome is the direct result of the nature of the IETF, and the particular personalities overseeing this work. To be clear, these are not bad or incompetent individuals. On the contrary – they are all very capable, bright, and otherwise pleasant. But most of them show up to serve their corporate overlords, and it's practically impossible for the rest of us to compete.

Bringing OAuth to the IETF was a huge mistake. Not that the alternative (WRAP) would have been a better outcome, but at least it would have taken three less years to figure that out. I stuck around as long as I could stand it, to fight for what I thought was best

for the web. I had nothing personally to gain from the decisions being made. At the end, one voice in opposition can slow things down, but can't make a difference.

Conclusion (by Nikos Leoutsarakos)

Eran wrote these prophetic words about eminent trouble with OAuth 2.0 in July of 2012 and less than a year later Egor Homakov described Open Redirect and soon after Wang Jing talked about Covert Redirect, two attacks that take advantage of the same inherent OAuth 2.0 vulnerability, just as Eran predicted (http://tetraph.com/covert_redirect/oauth2_openid_covert_redirect.html).

I have three objections to how OAuth and OpenID authenticate users that Eran also hinted. First, these technologies require a third party authenticator, a worldwide authentication authority which is highly unlikely to be accepted worldwide. Second, these technologies provide security by tokens that Websites receive from the authentication authority, and it is only time before these tokens get altered and misused. And third, these technologies are "sentinel" processes that can be easily bypassed, i.e. no need to break them in order to get to the digital valuables they protect.

To all those "lazy" Websites that rely on Facebook to login your users, let this be a lesson and please do it for your users: stop using OAuth 2.0 and OpenID.