hueniverse

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 (http://tools.ietf.org/html/draft-ietf-oauth-v2).

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.

[illegible]e wasn't a single problem or incident I can point to in order to explain such an extreme [illegible]e. This is a case of death by a thousand cuts, and as the work was winding down, I've [illegible]d 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 (http://tools.ietf.org/html/draft-ietf-oauth-v2-30#section-1.8).

When compared with OAuth 1.0 (http://tools.ietf.org/html/rfc5849), 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 hand of a developer with deep understanding of web security will likely result is 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 (http://hueniverse.com/2010/09/oauth-bearer-tokens-are-a-terrible-idea/). Any improvement in token security requires additional specifications and as the current proposals demonstrate (http://openid.net/specs/draft-jones-json-web-token-07.html), 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 issues 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 the 2.0 threat model (http://tools.ietf.org/html/draft-ietf-oauth-v2-threatmodel) 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 (http://openid.net/connect/) (which too was a super simple proposal 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

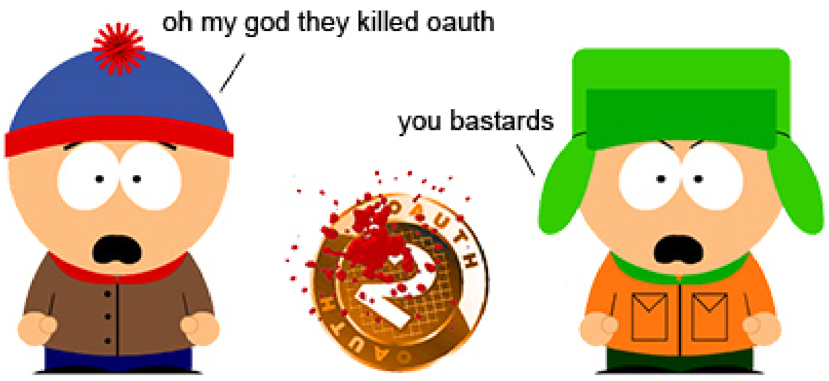
This is a sad conclusion to a once promising community. OAuth was the poster child of small, quick, and useful standards, produced outside standards bodies without all the process and legal overhead.

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 (http://hueniverse.com/2009/11/wrap-and-the-demise-of-the-oauth-community/), 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.

I failed.

We failed.



(http://hueniversedotcom.files.wordpress.com/2012/07/oauthdead.jpg)

Some more thoughts… (http://hueniverse.com/2012/07/on-leaving-oauth/)

□ July 26, 2012 □ Eran Hammer

153 thoughts on "OAuth 2.0 and the Road to Hell"

1. Pingback: Entwicklungsleiter distanziert sich von OAuth 2.0 ~
2. Pingback: Hoofdontwikkelaar stopt uit onvrede met OAuth 2.0-specificatie | Trending in Nederland ~

3. Pingback: Entwickler: "OAuth 2.0 weniger interoperabel und weniger sicher" | Edv-Sicherheitskonzepte.de – News Blog aus vielen Bereichen ~
4. *Torsten Lodderstedt* says:
   July 29, 2012 at 7:41 am
   Hi Eran,

   as already indicated on the list. I think you made a great job as editor of the base spec. Sadly, your and mine assessment of what we have achieved so far seem to contradict. At Deutsche Telekom, we have implemented proprietary token protocols as well as OAuth 1 and 2. Based on our experiences I would conclude, OAuth 2 is by no means perfect but better in terms of interoperability, usablility, and scalability than everything else we had before. And our security team is happy as well.

   As editor of the OAuth 2.0 security document I would be eager to know how you came to the conclusion OAuth 2 is less secure than OAuth 1. What problems of practical relevance do you see? And what evidences do you have?

5. *Joseph Werle* says:
   July 29, 2012 at 9:30 am
   This was beyond inspiring and shows how one can truly be humble about a project they started and knowing when to throw in the towel. It is another spec gone wild and to far out to come back. Being the creator and recognizing that is truly a power many don't have..

6. *Mojo Jojo* says:
   July 29, 2012 at 9:44 am
   The 2.0 draft is roughly double the number of pages of 1.0.

   Double the page count for a revision is usually a sign of administrivium and committee-itis, of anything other than "a revision of an existing protocol".

7. *Bob Denny* says:
   July 29, 2012 at 5:34 pm
   This scenario has been repeated ad nauseum as long as I have been associated with computing and communications (my whole life, 40+ years as a developer and I am current and still do it full time as a profession). Having been through a similar evolution in the area of a protocol for distributing transient astronomical events for followup, I can say that the collective farming model of engineering is flourishing. Under this model, people are more concerned with form than results, and everyone has to urinate on the fire hydrant. Compare with engineering where elegance, results, prototyping, and incremental refinement in the real world, are king. Reading your story was a trip down many memory lanes. I'm sorry for your loss.

8. Pingback: Eran bails out of the OAuth2.0 Spec at "Find Thomas on the Net" . . . ~
9. *Eran Hammer* says:
   July 29, 2012 at 10:32 pm
   Worth reading: http://www.tbray.org/ongoing/When/201x/2012/07/28/Oauth2-dead

10. Pingback: OAuth 2.0 standard editor quits, takes name off spec • The Register - Techbeast.net ~
11. Pingback: OAuth 2.0 and the Road to Hell ~

12. Pingback: Ontwikkelaar: 'OAuth is de weg naar de hel' | Tech-nieuws ~
13. *James Henstridge* says:
    July 30, 2012 at 8:23 am
    Looking through your complaints, could you elaborate why the "unbounded tokens" issue is bad? It isn't clear what benefit there is to the client in providing its ID in every request, and the server obviously knows who it issued the token to (and could encode the client ID in the token if it really wants). And the OAuth 2 design sounds like it would let the service limit knowledge of client secrets to the code that issues tokens rather than every piece of code that checks signatures needing those secrets.

    As for token types, OAuth 1 has PLAINTEXT and HMAC-SHA1, while OAuth 2 has bearer and HMAC token types. While there are obviously places where OAuth 1 HMAC-SHA1 signatures are suitable where OAuth 2 bearer tokens aren't, that isn't really a fair comparison. Comparing PLAINTEXT signatures with bearer tokens makes more sense, and I think bearer tokens come out ahead. They do away with unnnecessary timestamps and nonces, which only add failure modes rather than extra security (e.g. does it really matter if a client's clock is wrong when they're using PLAINTEXT auth?). Reserving that complexity for HMAC tokens seems like a sensible improvement.

    As for token expiration, OAuth 1 clients need to deal with the possibility that their token may become invalid some time down the track (possibly because it expired). It isn't obvious to me that formalising this case is a bad thing.

    I haven't read the grants section in detail, but I will say that I've seen people get confused about the distinction between OAuth 1 request tokens and access tokens. So using different terminology for these different phases of the authorisation process seems somewhat sane. Documeinting a method to implement "desktop username/password auth" (not via web browser) also seems like a decent addition too. When we wanted to do this with OAuth 1, the spec offered no guidance but we ended up with something that is roughly equivalent.

    I haven't used OAuth 2 in anger yet, so perhaps I am wrong about some of this. But from my use of OAuth 1, the changes don't all obviously sound like bad ideas.

    - *Eran Hammer* says:
      July 30, 2012 at 9:00 am
      The changes are all valid technical choices. They are not necessarily bad on their own. What they do is make OAuth better at scale by stripping down protections under the assumption that other layers will take care of things. The problem is, this is not how modern security protocols should be designed with layered security. Because the token is unbounded, all you need is to steal just one thing, and is TLS wasn't configured correctly, you're toast. Also, with bounded tokens, a client must share its own credentials to share tokens around. This is a good protection against an approved application leaking your data.

      - *James Henstridge* says:
        July 30, 2012 at 7:19 pm
        I realise that if the transport layer security fails and a bearer token is leaked, it can be used as credentials for other requests. But the same is true for OAuth 1 PLAINTEXT signatures: while you need four values to generate the signature instead of just one,

those four are always presented together so it isn't clear they offer any more security than a single value. If anything, it looks like the bearer token would offer slightly more security since such a leak won't expose the client ID and secret.

Now of course OAuth 1's HMAC-SHA1 signatures offer better security in this situation. But that point seems moot given that the OAuth 2 specs document HMAC tokens. I realise that HMAC tokens only use two values to construct the signature while OAuth 1's signatures use four, but those four values in OAuth 1 signatures may as well only be two given the way they're used (consumer_key+token and consumer_secret+token_secret).

It really doesn't look any worse than the situation of OAuth 1 documenting multiple signing algorithms.

- *Daniel Friesen* says:
  July 30, 2012 at 7:40 pm
  Btw, side note reading the OAuth 2 MAC spec and OAuth 1's HMAC-SHA1 section I noticed something.
  OAuth 1 signatures include the POST body for form-urlencoded data while OAuth 2 MACs offer no protection to the entity-body.

- *Eran Hammer* says:
  July 30, 2012 at 10:57 pm
  True. It ended being an encoding/decoding nightmare. Instead, MAC offered an ext parameter for stuff like body hash that's application specific.

- *Daniel Friesen* says:
  July 30, 2012 at 11:38 pm
  Yeah… the way decoded query parameters was worked with and then encoded a specific way in a specific order was a mess.

  I've been contemplating how I'd write a spec like this.
  I looked over some of the scripting languages. It looks like practically every language seems to give the script access to the raw post body (with the exception of things like multipart/form-data which you would stream and wouldn't want a signature for).

  Under that vein my thought was to have two modes for the mac. One that excludes the body and another that includes the body in the signature (or rather includes a hash of the body into the signature, so large data can be verified after processing without requiring double storage of the original data).

  A client would decide what mode to work in depending on the data they are working with. Large contents and multipart documents you'd use the mode without the body. And with form-urlencoded, json, xml, etc… post APIs you would sign the body.

- *Daniel Friesen* says:
  July 30, 2012 at 8:11 pm

OAuth 1 emphasized signatures while OAuth 2 seems to emphasize bearer tokens. So there is a an issue with perception of the spec to consider. Everyone who used OAuth 1 used signatures. While everyone who has used OAuth 2 have only used bearer tokens and seem to consider this security degradation as and advantage.

That aside technically yes we should probably compare OAuth 1 HMAC to OAuth 2 MAC tokens.

On the topic of unbounded tokens combined with (H)MAC signatures.
Assuming the TLS connection fails (the client developer decides to disable it, etc…) doesn't the fact that the access token is unbounded leave it open to abuse by the MITM?

In an OAuth 1 signature the client secret and token secret are used together. While an OAuth 2 signature just uses the token secret.
Because HMAC is used the token secret is safe in both cases for requests using the token.

But what about the token endpoint itself? If TLS protection has failed then isn't it likely that the MITM is capable of watching you make the grant request to the token endpoint asking for the token?
So even though we're using signatures doesn't this mean that the MITM has likely gotten ahold of the MAC token and knows the secret already?

OAuth 1 uses the client secret and the token secret. So you shouldn't be able to do anything malicious since the MITM doesn't know the client secret. But OAuth 2 doesn't use the client credentials anywhere (until a refresh) after the mac credentials are handed over.

Doesn't this mean that OAuth 1 bounded tokens protect the resource endpoint from attack while OAuth 2 unbounded tokens allow a MITM to freely abuse the access token maliciously?

- *James Henstridge* says:
  July 30, 2012 at 10:14 pm
  In the current draft of the OAuth 2 spec, token types are introduced in Section 7.1. It mentions bearer tokens with a link to the bearer token spec, and then mentions MAC tokens in the very next paragraph along with a link to that spec. So are services picking OAuth 2 bearer tokens that different to ones picking OAuth 1 PLAINTEXT signatures?

  My point wasn't that bearer tokens are safe against TLS failures, but rather that they are no worse than the OAuth 1 PLAINTEXT signatures. Now if there are defficiencies in OAuth 2 MAC tokens, that would seem to be a separate matter.

  It does seem that MAC token authentication doesn't protect the request body. It looks like the first draft included a body hash in the signed request string, but was removed in the current draft. I'm not sure why that was changed, but it looks like a service could reintroduce it through the "ext" parameter if it chose to. That's not particularly great for interoperability and not a fatal problem.

The other issue is that the entire MAC key is sent over the wire when the token is issued (a problem if the TLS fails and the connection is intercepted). I agree that OAuth 1 has a leg up here, since a portion of the MAC key is never transmitted over the wire (the client secret)..

Surely it would be better to concentrate on these issues directly rather than making strawman comparisons with bearer tokens.

- *Eran Hammer* says:
  July 30, 2012 at 10:54 pm
  First, very few providers implemented plaintext. OAuth 1.0 gave them three options: light, medium, and heavy. It is true that if you capture a plaintext request it is game over, but there are many other scenarios other then the ability to listen to the entire channel. In 1.0, you provision a token to a specific client, and they you enforce that restriction. You can't do that in 2.0 as specified. As to HMAC in 2.0, I've been trying to get it supported for three year but at this point cannot predict if the working group will finish that effort. Less than 5 people showed interest last time I checked.

14. Pingback: Dare Obasanjo aka Carnage4Life - OAuth 2.0: The good, the bad and the ugly ~
15. Pingback: Why I Still Like OAuth | K. Scott Morrison's Blog ~
16. *Rosco* says:
    July 30, 2012 at 4:05 pm
    You could work on a new spec, call it NAuth for "New Auth – one step ahead of OAuth"!

17. Pingback: Software Maniacs blog » OAuth is not a protocol ~
18. Pingback: Revue de Presse Xebia | Blog Xebia France ~
19. *SV* says:
    July 31, 2012 at 6:39 am
    This reminds me of the XSLT/XPATH 2.0 debacle at the W3C. James Clark, the editor and driving force behind XSLT/XPATH 1.0 dropped out of the effort early on – I and my colleague at Novell hung in there for several years before dropping out when it became clear that it was going to be several more years before it would be done and that the result would be way several orders of magnitude more complex than 1.0, which was already very difficult for many of our constituency to understand.

20. Pingback: OAuth: Another Spec Bites the Dust | Cigital ~
21. *Egor Homakov* says:
    August 1, 2012 at 3:18 am
    Eran, this: http://homakov.blogspot.com/2012/08/saferweb-oauth2a-or-lets-just-fix-it.html

22. Pingback: » Authentication: Why can't we solve this age-old irritation? Tom Russell – A Ruby/Rails Developer ~
23. Pingback: Awful OAuth 2.0? When Standards go bad. – Simple-Talk ~
24. Pingback: OAut(sc)h - notizBlog ~
25. *Jeff Brandt* says:
    August 3, 2012 at 1:22 pm
    Great article but I hate to hear the news. I attended a Healtcare meeting on a proposal to use OAuth2 and it sound like overkill.

It reminds me of all my work in the 90's on SET with IBM (Secure Electronic Transactions), a very heavy secure protocol. All of a sudden SSL came on the table and SET died. SSL was "good enough"

Best of luck,
Jeff

- *KatieP* says:
  August 6, 2012 at 9:13 am
  Yes, that seems to be the case. OAuth is designed to solve a simple problem in a complicated universe. And there were many complicated tries on this WS-*, SAML (Federation).. I put this down as the growing pain of this simple protocol.

26. *peter williams* says:
    August 8, 2012 at 8:05 am
    Again, the overall tone of the comments is correct. The criticism is IDENTICAL to the criticism levied at X.509. X.509 in its v3 incarnation was a framework – of a 1000 incompatibilities and a notation for easily manufacturing more.

    Of course, eran also notes the solution to a framework-notational spec (which occured to X.509, another infamous framework-notational spec). Someone comes along and makes a super-dominant profile, probably webby, that is so compelling that everyone else needs to connect. For X.509, this was of course the netscape SSL server certificate. Only in windows do you see, enterprise-like the 24 other profiles that hardl anyone anyone ever sees (and actually rarely bitch about).

    NOw the number of of folks who bitch about X.509 is larger by 2 orders of magnitude than those who bitch at OAUTH v2 – mostly because its just been around for 20 years longer and the bitch-fests just get more political, more religious, and more hatred filled each year – as function of _reach_ (didnt you know X.509 today is still an ISO/Soviet/AlQueada plot to take over the US and impose 8bit micros on the world, rather popular in 1986 when X.509 first came about?) But also note, WHEN THE FRAMEWORK IS RIGHT, you are stilling talking about a silly bit format 20 (yes 20) years later.

    Thats what standards are for. They are there to be still around 20 and for X.509 30 years later, having evolved piecemeal for the last 15 of them using the notational-framework – evolving subtly and dinosaur like as more and more economic dependence is placed on their shoulders.

27. *Raymond Forbes* says:
    August 8, 2012 at 10:14 pm
    yes yes and more yes! thanks so much for posting this. i watched the oauth 2.0 spec being developed after learning oauth 1.0 and i was vastly disappointed in the decisions that were being made. it really felt like they were sacrificing security for ease of use, which is never a good thing for a security spec.

    -r

28. Pingback: Is OAuth 2.0 sinking? | Health, Vitality and Technology ~
29. *Attaullah Baig* says:
    August 26, 2012 at 8:21 am

There was no such thing as “simplicity”

30. *Andres Felipe Diaz* says:
August 26, 2012 at 3:41 pm
Not much to add to all of this discussion.

    However I want to thank you for your time dedicated throughout these years. Your geek efforts have made a huge difference.

    The guide was also awesome and very well explanatory of how OAuth 1.0 works.

    Now…as Mr. Wayne would say…”Eran Hammer, why do we fall sir? So we might learn to pick ourselves up”

    I hope you keep producing valuable things for the community.

    Best,
    Andrés

31. Pingback: In Defense of OAuth 2.0 ~
32. Pingback: SharePoint 2013: What to do? Farm solution vs Sandbox vs App « SharePoint Dragons ~
33. Pingback: CJG : OAuth 2.0 - How long with it survive? ~
34. Pingback: The internet needs fewer customers, fewer products and more hippies. | andrewt.net ~
    - *Nils* says:
      September 11, 2012 at 6:27 am
      I’ve been a consultant for the best part of my life. I’ve been implementing access management solutions, SSO solutions, federated SSO and now I’m at the dawn of setting op a federated authorisation architecture for a big corporate organisation. I have not read all of these posts, but can somebody tell me what to do know. I need to provide the answer to the customer tomorrow…sic

Comments are closed.