UMA F2F 2009-11-02

UMA F2F 2009-11-02

Date and Time

- **Day:** Monday, 2 Nov 2009
- **Time:** 1:00-5:00pm PT (meeting held locally; no dial-in)
- **Location:** Boole room, Computer History Museum, Mountain View, CA (directions)

Attendees

Quorum is 12 of 22 (due to two signups during the meeting!).

Voting participants:

1. Akram, Hasan
2. Andrieu, Joe
3. Bryan, Paul
4. Curran, Sam (NEW)
5. Davis, Peter
6. Hanson, Michael
7. Holodnik, Tom
8. Machulak, Maciej
9. Maler, Eve
10. Smith, Bill
11. Stollman, Jeff
12. Windley, Phil (NEW)

Regrets

- George Fletcher
- Domenico Catalano
- Gregg Kellogg

Agenda

- Roll call and administrative stuff
    - "IPR benediction"
    - Shall we use #umaf2f along with #iiw as appropriate for tweets, pics, etc.?
    - Approve minutes of UMA telecon 2009-10-29
- Action item review
- Spec review
    - Latest: email
- Scenario development (docket)
    - Focus on scenarios and use cases involving user-driven terms, policies, and demands for claims
    - Understand where the IMI model of claim requests and responses may be able to help us
- AOB

Minutes

Roll call and administrative stuff

Quorum reached.

- "IPR benediction"

People attending and contributing to this meeting need to sign the GPA.

- Shall we use #umaf2f along with #iiw as appropriate for tweets, pics, etc.?

Yes.

- Approve minutes of UMA telecon 2009-10-29

Minutes APPROVED.

Action item review

Nothing has changed since Thursday.

Spec review

- Latest: email

We looked more closely at Step 3, which we know needs to change. Paul would like the host to get out of the business of deeply managing a requester's authorization, only being involved for correlation all the way through to the AM. Can the host create an "authorization resource" at the AM that the requester can then access?

If we do this, what is the right granularity/mapping for authorized resources to protected resources? The authorized resource could apply to a single protected resource for all its access methods, a single protected resource+access method (so, two authorized resources for calendar+GET and calendar+POST), all resources at that host for that requester, etc.

We want to avoid prematurely optimizing the answer here, and we want to ensure that we meet our simplicity goals particularly around the host endpoint, to increase adoption. Michael observes, looking at the OAuth session extension discussions, that we might want to keep the number of state tables on the host way down.

NEW emerging design principle: Complexity: Complexity should be borne by the AM endpoint vs. the host or requester, if possible. Comment: We anticipate dozens of AMs (maybe lots more if corporations have them), hundreds of thousands of hosts, and hundreds of millions of requesters.

In particular, Paul observes that ProtectServe had been turning into a kind of authentication protocol in order to achieve optimization, and we don't want to go there.

Do we need to *profile* OAuth specifically for use as our authentication substrate?

Note that there are two kinds of "two-legged OAuth": unique consumer keys, and unique access tokens. Google currently uses the former. (The access token has always been an optional parameter of OAuth.) But this pushes out-of-band the issuance of consumer keys and secrets. The other option is using an anonymous consumer key to negotiate a special access token. This is the moral equivalent of a person doing "self-registration" at a website. There's no identity assurance, just correlation going forward. The secret that backs the access token becomes a shared secret. So it's like automated self-registration, which is kind of cool.

Facebook currently has a per-application secret, and per-user privacy settings, but no way for a new application to approach Facebook Connect dynamically.

Where a requester is going to an AM for the first time, it needs to establish an identifier for use in future direct requests made to the AM.

Eve is a little worried that using "nonstandard" OAuth could be a barrier to developer adoption. But if we provision the consumer key right in the XRD, maybe the way it would be used ends up being pretty much like "regular" OAuth.

So Paul's current idea is that the requester asks the host for a "referral" over to the AM. This got us into a discussion of how he's proposed to use XRD to convey the challenge rather than in a 403 header. There is a variety of ways the XRD location can be indicated. Michael noted that on the webfinger list, there's been discussion on the propriety of subdomain XRDs, with Eran speaking against it. Since the same party might serve as a host for more than one user, to be truly "user-managed" it needs to have multiple XRDs.

If we had the freedom to invent another www-authenticate header, it would solve a lot. We're not sure yet if that's kosher.

If we have a whole bunch of XRDs, we're going to have to explain to potential hosts how they need to be prepared to serve up XRDs for their users if they want to take advantage of being a PEP.

In choosing between an XRD mechanism and a challenge-header mechanism, one consideration is that in the latter, you're declaring that info to be the "one true way" to achieve authorization. Experience shows that whatever method is last in a chain of such methods is the one that gets used, which is not necessarily the desired effect.

We went through a swimlane diagram exercise (click the link to see the resulting diagram).

The requester could forestall the beginning part of the process by getting the XRD some other way, before the 403 or even before the 401 (not shown) that preceded it.

Referrals are transient; their only purpose is to correlate a host's view of a requester's identifier with the AM's view of that same requester. Once the link is made, the referral stuff can go away. (The host and AM can happily manage that state separately afterwards.) The ID provided by the host can be the user ID, the session ID, or whatever is handy.

ISSUE for later: How and whether to request the "protected status" of a resource: To answer this question, a host would pretty much have to go through the same dance as for a request for actual access! It might be protected against requester A but not requester B, or protected with a real-time user consent loop, etc. We'll wait to see what real scenarios arise that need to be solved, and perhaps it will turn out that they can tolerate imprecision/latency. Before a host gives out the 401, it already knows intrinsically whether the resource is protected by some AM, and it has the *ability* to tell the requester this (whether this is a good idea or not we don't know yet). It just doesn't yet know whether the requester is going to be authorized to access it.

Since we're now out of the authentication business, keep in mind that whatever the host does in response to the initial approach of the requester is up to it. It can assign a pseudonymous form of identifier (possibly literally using a cookie-based method), or really anything. If the host needs to protect the privacy of the requester according to its own policies or applicable laws or whatever, it's up to the host to choose an ID wisely. We don't really care about protecting the requester's privacy, however; the whole point of letting the authorizing user control access is to let them do so on whatever criteria, and as part of that process the requester is simply going to have to authenticate, even if weakly.

OAuth (the subject of our design principle #2) actually brings two different use cases to the table: service authentication and a "read/write" framework for service access. Both are valuable to us.

NEW emerging design principle: Authentication: We want to stay out of the authentication business as much as we can. Comment: There are many technology choices here, and we want to be agnostic as to the right one for the job. Some scenarios may need stronger authentication and they should be free to use the right mechanism for the job. And setting up all pairwise relationships may be subject to some trust framework (e.g., some AMs might get blacklisted).

In the two-party introductions, we have the same vulnerability as OAuth around spoofed (not really phishing) sites. SSL/TLS mitigates this, as usual.

Scenario development (docket)

- Focus on scenarios and use cases involving user-driven terms, policies, and demands for claims
- Understand where the IMI model of claim requests and responses may be able to help us

We walked through Eve's screen mockup (see slide 8 that shows how policies and terms might be set, and brainstormed what features the protocol might need to support.

The box on the top represents a set of (presumably standardized but currently totally made-up) contract **terms** that a user might want to offer a requester, such that if the requester *meets the terms* (which might involve agreeing, paying, etc.) they will get authorized for access.

The box on the bottom left represents a length-of-access **policy** that the user might want to set, which the AM can act on unilaterally (without requester input). We discussed whether the "Once (unlimited tries till success)" option is practical; for non-GET types of service access it is practical (and, in fact, it's a requirement that we need to satisfy), but for GET types of access it's not practical because of the nature of HTTP. The actual option for length of access in a "one-time" GET case would probably be something like "as many times as you like in the 10 minutes (or hour etc.) after your first attempt at access". The user experience shouldn't expose this complexity, however! It could say "once" as a first approximation, and offer a link for an advanced explanation of how it really works.

NEW pending requirement: Something about ensuring that it's possible for AMs to offer a "POST once" setting, as this is critical for payments and the like.

It was suggested that we should offer UX best practices for policy, as OAuth and OpenID do. Whatever policies and terms we highlight in such materials will be likelier to be adopted. As well, it was suggested that we should provide examples of how to solve for personal-datastore types of basic data.

The box on the bottom right represents what we anticipate to be a likely case in some scenarios (such as "hey, sailor", where the user widely advertises a link that's UMA-protected): the authorizing user wants the AM to contact him when the authorization request comes in, asking for real-time consent. It's possible that this *policy* also has consequences for *terms*, in that the requester probably should be informed (or even be asked to agree with the fact that) the user may choose not to consent even if the requester has jumped through some hoops already to get authorized.

It was suggested that a condition people might want to make for access is "I'll offer you access to this, if you offer similar access to your stuff." This might be especially important for more peer-to-peer (person-to-person) types of sharing; we invite scenarios on this.

The Information Sharing WG's car-buying scenario highlights the potential need for data-sharing terms to have some specificity to the type of data being shared.

NEW issue: Do we need a new DP for "value of access"? It should range from low to high, which may impact security options chosen.

We need to do some serious scenario-building to guide our work on terms negotiation. Here are some ideas we brainstormed, with nicknames:

Name of Potential Terms Scenario	Description
Always Authorize, or Just Audit, or Requester Whitelist	AM->R: (nothing). R->AM: yippee! I can get in!
Requester Agrees	AM->R: Here are the authorizing user's terms; accept them. R->AM: I agree.
Requester Supplies Policy	AM->R: Tell me your privacy and data usage policy. R->AM: Here it is.
Porn Uploading	AM->R: Prove to me you're over 18. R->AM: (claim).
Contract Exchange (a la Nat's work)	AM->R: Make me an offer. R->AM: (terms). (potentially cycles back and forth a few times after this, with AM/user choosing some terms over others)
But Wait, There's More	AM-R: I'll give you resource A if you meet terms X, but also resource B if you agree to terms Y. R->AM: (claims meeting terms X or Y)
Boolean	...

Next Meeting: UMA telecon 2009-11-12

NOTE: The UTC hour has changed, but we're back on our "normal" schedule (9am in Seattle, 6pm in Munich, and so on) now that all seasonal time changes have been taken into account worldwide.

- **Day:** Thursday, 12 Nov 2009
- **Time:** 9:00-10:30am PDT | 12:00-1:30pm EDT | 17:00-18:30 UTC (time chart)
- **Dial-In:**
  - Skype: +9900827042954214
  - US: +1-201-793-9022 | Room Code: 295-4214 (other local country numbers available on request)