location_scenario

Scenario: Controlling Two-Way Sharing of Location Information (Pending)

Submitted by: Eve Maler

Actors

- Authorizing user Alice, who chooses to set and share her current location through various applications that she accesses with a commercial off-the-shelf browser
- An application that serves as an **Authorization Manager** (AM) on Alice's behalf, orchestrating which location applications can write to and read from other applications
- Web-based location applications called **HotLocale** (H1) and **HipHappeningPlaces** (H2) that are Hosts of location information on Alice's behalf
- Web-based location applications called **RovingAround** (R1) and **RoadWarrior** (R2) that are Requesters of location information on Alice's behalf

Short Description

Today's location services such as FireEagle, BrightKite, and Dopplr let Alice set her location within any one of several applications, and then use OAuth-enabled connections to propagate that information through other such services. Since Alice can end up "chaining" services this way quite easily, with a thicket of pairwise connections, it's valuable for her to know and control where this information is flowing to. In this scenario, a user of HotLocale and RovingRound wants to arrange to connect them up so that her location can be propagated among them, and she wants to get a global view through her AM about who's allowed to do what, so she can change and stop permissions in a coordinated way.

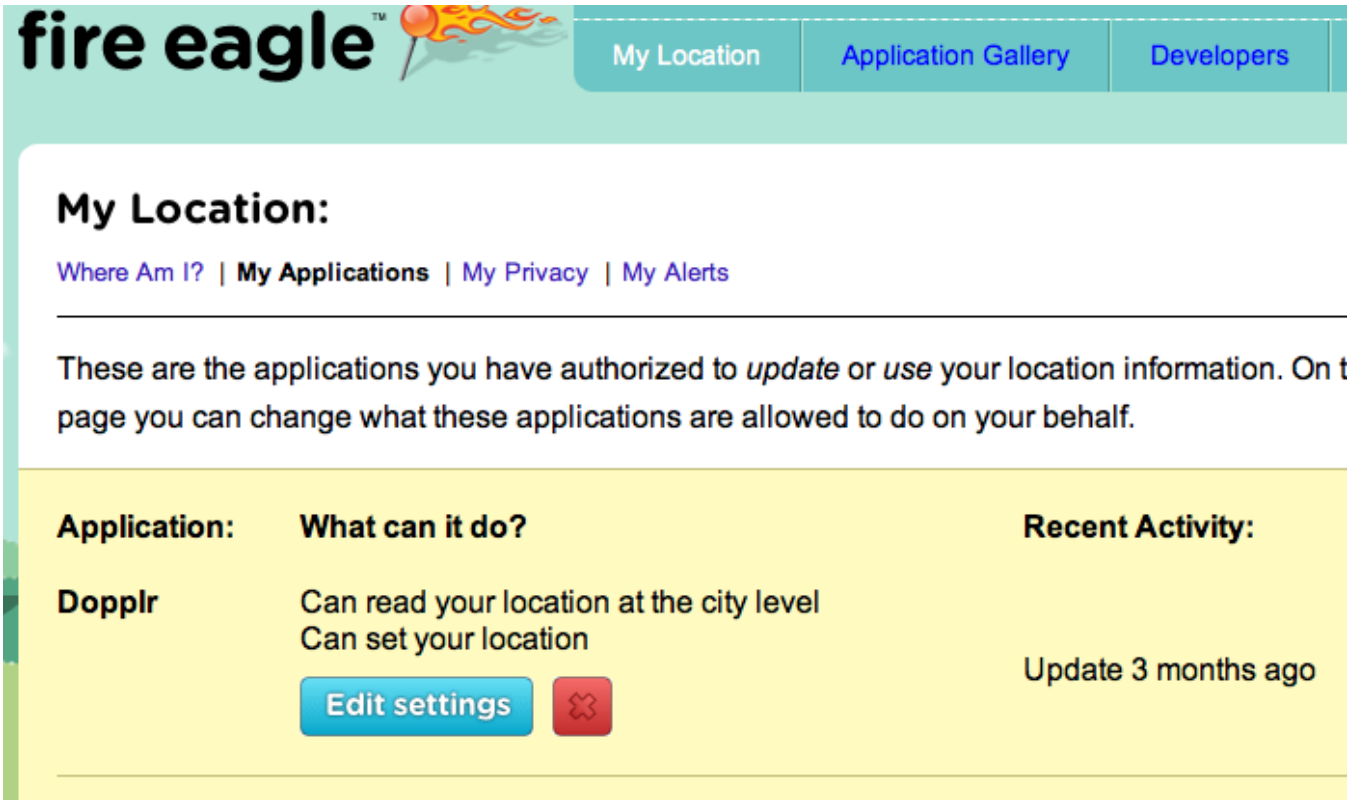
Note: This scenario is not exploring anything other than person-to-self sharing. How Alice exposes her location to other people and companies should be the subject of a different scenario, as warranted.

Dimensions

Nature of protected resource	API endpoint. For example, see the FireEagle developer documentation. The resource being protected is a single endpoint URL, with different options and parameters possible for using it. The possible scopes in today's location services tend to include (a) whether the requesting app (client) can write the user's location, and (b) to what degree the client can read the user's location. For example, see the FireEagle scope permissions documentation; the location-reading options include: no read access, as precise as possible, postal code level, neighborhood level, town level, regional level, state level, and country level.
Sharing models	Current OAuth-protected usage of location services such as FireEagle and BrightKite assumes a **person-to-self** sharing model, where Alice has login accounts at the various host and requester apps, and wants to instruct them to share her location among themselves on her own behalf; an UMA-enabled version of this would allow her to control such sharing centrally at her AM.
Nature of policies and claims	For person-to-self sharing, a typical policy might require the requesting party to be able to authenticate as Alice directly at the AM (that is, dictating a particular Step 2 web-server flow that uses normal OAuth user authentication), or might require a claim that it is an authenticated Alice asking for forge the connection. Further, a policy might set (unilateral) boundaries on which scopes can be accessed.
Cardinality	This scenario is 1:1. There is no need to aggregate multiple resources from multiple hosts, for example.
Colocation	The same requester app might both read and write location data (by using different methods with the same API endpoint), which allows it to *write* data to a host while still remaining a *requester* (think "requester of API access"). With today's OAuth-enabled location services, the services act relatively "peer"-ish, in that they have each done the necessary integration to support any one of them serving as an initial host with the others being requesters. So it could be said that the same app might be a host for some users and a requester for others, and/or might be set up as both a host and a requester for the same user's location; however, the latter may be very confusing given current OAuth realities.
Host-AM relationship	Today's OAuth-enabled apps require static introduction and configuration, but the light weight and low security of these Web 2.0 services suggest that dynamic introduction is a possibility as long as the location API is well-known/standardized.
Protected resource discovery	Today's OAuth-enabled apps advertise this location in the process of static introduction and configuration, but with a well-known/standardized location API, it seems possible that a location service host could advertise its endpoint through dynamic means such hostmeta/XRD.

Scope Detail

Below is a screenshot showing that FireEagle and Dopplr have the capability today to have two-way location information flow. Our user wants to be able to see this "combinatorily", for all connected location services and indeed for all such services on the web that she chooses to use for hosting any data or content.



Assumptions and Preconditions

This scenario assumes that Alice has an account at each of AM, H1, H2, R1, and R2. These accounts might or might not be driven off of federated login, for example, Alice might log in to HotLocale through Google and RoadWarrior through Facebook.

This scenario assumes that AM, H1, H2, R1, and R2 are all UMA-enabled but have otherwise not met before. (Simplified circumstances that assume prior introduction are described in their turn below.)

This scenario assumes that H1 and H2 use a hypothetical standard location API that R1 and R2 are configured to understand, and that H1 and H2 expose APIs through URLs that differ per user in some fashion (e.g., through a URL query parameter or through a part of the URL path).

Use Case 1: Alice Sets Up AM Protection Over Location Information at HotLocale

This flow can be embedded in other flows, or can be standalone.

1. H1 asks Alice to choose an AM to protect her location information in H1
2. Alice tells H1 her preferred AM
3. H1 discovers how to get started connecting to the AM
4. Alice is redirected to AM to log in as Alice-on-AM, consent to protection, and choose a policy that applies to H1's endpoint for her
    - The policy indicates that the requesting party must accept her chosen scope of access, which is "write, read-city"
    - The policy indicates that the requesting party must be able to log in at the AM synchronously as Alice-on-AM and consent
    - The policy does not otherwise discriminate against particular Requester apps (*note that we don't currently have a way to do this; do we need to?*)
    - She labels the policy "location services policy" (outside the scope of the UMA protocol)
5. Alice is redirected back to H1 to continue what she was doing before

The identical sequence can be done with HipHappeningPlaces (H2).

Unknown macro: 'html'

Use Case 2: Alice Shares HotLocale Location Access with RovingAround and RoadWarrior

1. Alice visits H1 and logs in as Alice-on-H1
2. Alice decides to share access to her location information in H1 with R1 (if use case #1 has not been done, it must be done at this time)
3. Alice asks H1 to give her a "share" URL to use
4. Alice visits R1 and logs in as Alice-on-R1
5. Alice gives R1 the "share" URL from H1 and asks it to set up access to her location information there
6. R1 attempts to access the URL and discovers H1 uses a location API that R1 understands
7. H1 redirects R1 to the AM as unauthorized

8. AM tells R1 it needs to provide a claim acknowledging the offered scope (*need to standardize this claim request/response*)
9. R1 sends the claim to AM
10. AM tells R1 it needs to redirect its user (Alice) to AM to authenticate (*add this as a special indicator/claim request to the protocol?*)
11. R1 redirects Alice to AM to authenticate and consent
12. AM concludes that Alice authenticated correctly as Alice-on-AM
13. AM issues an access token to R1 for location information access of the offered/accepted scope and redirects Alice back

Unknown macro: 'html'

The identical sequence can be done with RoadWarrior (R2), except that once HotLocale is introduced to the AM, it never needs to be introduced again, so the optional embedded UC1 block isn't ever done again unless Alice wants to switch AMs.

Use Case 3: Alice Monitors and Controls Location Information Access from Her AM

To monitor access, Alice interacts with value-added functionality provided at the "data-sharing relationship manager" application that serves as her AM endpoint; that is, this use case does not involve standardized UMA protocol behavior. These interactions might include:

- Looking at a unified log for occasions when both RovingAround and RoadWarrior accessed HotLocale on her behalf, and what they did on those occasions
- Looking at a unified log for occasions when RovingAround accessed both HotLocale and Alice's calendar application (not part of this scenario) on her behalf
- Looking at a unified log for all hosts to whose resources she applied her "location services policy", including both HotLocale and HipHappeningPlaces
- Showing a graphical depiction of location host and requester access relationships and their types, if it is able to characterize hosts according to the APIs/resource types they expose

To control access, Alice may take various actions:

- If Alice discovers that her HotLocale account was hacked, she can go into her AM and revoke access to it from all requesters. If access tokens issued to requesters are sufficiently short-lived, the next time any of them seeks access by presenting a refresh token in order to get a new access token, they would be denied access. (*would the OAuth 2.0 token revocation extension help here?*)

- Alice may decide that she wants to limit access to include reading her location from HotLocale only, instead of reading and writing her location, because the previous situation was too confusing. She can change the "location services policy" to limit the scope list. If access tokens issued to requesters are sufficiently short-lived, the next time any of them seeks access by presenting a refresh token in order to get a new access token, they would be asked for a claim acknowledging the reduced scope and be issued a new access token.

Issues

- **Scope claim:** Need special claim request/response regarding some scope list being offered by the authorizing user, as imagined above?

- **RESTful API:** Do we really need the assumption that the protected resource URL has something special and authorizing-user-specific about it (with the invention of the "share URL" concept above)? Note that the share URL doesn't have to be a secret URL; it should be protected by UMA regardless (we'd need to think about how the host registers resources if we do make the "share URL" assumption). Is there some way we can handle current common API practice, where the endpoint URL is static and doesn't change per user?

- **Host API dynamicism:** Use case 2 could be called "host-initiated access"; this only works if the host uses a standard API that the requester can discover and work with. Do we need to build an alternative use case that assumes a proprietary API (akin to current OAuth usage), non-dynamic introduction of the parties, and requester-initiated access? How does scope work in that case – does the AM just end up recording the requester-offered scope?

- **Redirection of requesting user:** Do we need to bake a special claim request/response into UMA for handling the person-to-self flow (or at least usage of the web server user-authentication flow) in Step 2 so that the requester knows for sure to redirect the user to the AM? A requesting user who isn't Alice may not need to "authenticate" there, but they could (e.g.) provide a special code that was given to them by the authorizing user offline.