Integrating with the GPII Preferences Server

Meeting 15/1/16 to discuss UIO prefs server integration
Present:
Alan, Antranig, Avtar, Cindy, Giovanni, Justin, Michelle, Simon
Links from Cindy:
The work flow of using OAuth2 authorization code grant type: https://wiki.gpii.net/w/GPII_OAuth_2_Guide#Work_Flow
The wireframe for UIO and GPII integration: https://www.dropbox.com/s/v4pn0e4pzy1e4x4/uioptions%20GPII%20integration.pdf?dl=0
There are various OAuth 2 workflows available (Simon outlined 4). The most typical profile involves the use of an authorisation code, and secure server to server communication.
Early task:
Take an inventory of all the sites we have on which UIO might be deployed, and analyse what technologies are currently used, and how they are managed by ops.
Initial answer:
All of these sites are somewhat different. However, we plan to use the current deployment structure of the ILDH as a template for any future such sites and normalise them. ILDH currently uses nginx to serve up static content produced by docpad.
Worrying observation:
The requirements of the typical OAuth workflow mentioned above, appear to militate completely against the use of any kind of fully static site. What can we do about this?
Either we produce a standard "proxy" configuration that makes it as easy as possible to deform the configuration of a static site, OR we allow the use of alternative, less secure OAuth flows (implicit grant type) for this purpose, OR both.
Question:
What compromises are possible against the implicit flow variant - how easy are they, and what are their consequences?
Implicit grant type threats: http://tools.ietf.org/html/rfc6819#section-4.4.2
If you are able to get someone to visit a site that is not as it claims to be, you can "phish" for their preferences. [This can be mitigated by using HTTPS and verifying the redirect URL]
Access token is sent in HTTP redirect URL query fragment -- this may be exposed in Browser history or in server logs.
Alan: Browsers these days retain a lot of history. Especially in a shared computing environment, this is an unacceptable risk.
Gio's suggestion:
Create a "multi-personality" central auth server proxy that can act on behalf of multiple (possibly static) "client" sites, and act via a proxy configuration to just expose the endpoints on the static clients necessary to mediate the oauth and prefs communication.
We would probably do this via nginx. We would need to place the HTTPS layer IN FRONT of this layer of proxying - otherwise we would be unable to inspect and interfere with traffic at the HTTP level, e.g. to add a custom header or URL field.
A common pattern is to "terminate at the edge" - where the load balancing and HTTPS termination occurs at the edge of the cluster/cloud. The suggestion is that this would be the point in our architecture where the "selective proxying" described above would occur - which then implies that the architectural impact of our "otherwise static sites" remains simply that of static sites.
This is a brilliant suggestion.
Question (Justin):
What are the implications of having the multi-personality server have central control over parts of the workflow that might impact UX with respect to logon? Especially with respect to sites that have some logon semantics of their own - such as Wordpress. Further question - do we currently host Wordpress behind an nginx layer? Answer - yes, we do.
We will need to constuct both nginx-level and Wordpress-plugin-level embodiments of clients of the multi-personality auth proxy server.

Tasks:

1. Draw up the client's portion of the protocol in terms of the URL endpoints that it expects to operate.
The existing FD server is not a prototype for this since it uses the "client credentials" workflow type.
The existing EASIT integration did this - we are not sure where this is or what it looks like.
We will meet to address TASK ONE at 12 EST on Monday.
2. Figure out what needs to be added to this protocol by the "edge proxy" server to provide sufficient information to the "multi-personality" proxy.
Gio: We need to make sure that the edge proxy can be conversationally stateless. Anything stored at the edge should be static info e.g. per client website secret
3. Implement nginx configuration that implements the edge proxy, and built it into (Vagrant script?) that produces a standard Docker container embodying it in a configuration-driven way. It appears that this can be done purely within nginx' configuration system and will not require any custom plugins or code.
Gio: This may one day GO AWAY and implemented in KUBERNEETEES or such similar insane thing.
4. Design and implement the "multi-personality" proxy server - together with its own Vagrant/Docker configuration etc. This will be a standard config-driven Kettle server.

Flow #1 - Authorization Code Variant
1) User starts on website that has access to their preferences
2) They indicate they want to share their preferences
3) That website redirects them to a UI provided by the authorization server, together with a code that identifies the website that they came from
4) The authorization server, once the user has identified themselves, asks if the user wants to give access
- we know the identifier
- we redirect the user back to the site that they came from saying "the user will authorize you"
5) Authorization code is sent authorization code + shared secret

Flow #2 - Implicit Grant Variant
(incomplete - we will not pursue this in the first instance)