iLabs Authentication Proposal

Introduction
Currently the iLabs Service Broker (SB) has no authentication (AuthN) integration capabilities and performs local AuthN against an internal database independently of any prior AuthN the user may have already performed in another system such as a Learning Management System (LMS). This is shown in Fig 1.

1. User logs into LMS
2. url directs user to SB
3. User logs into SB

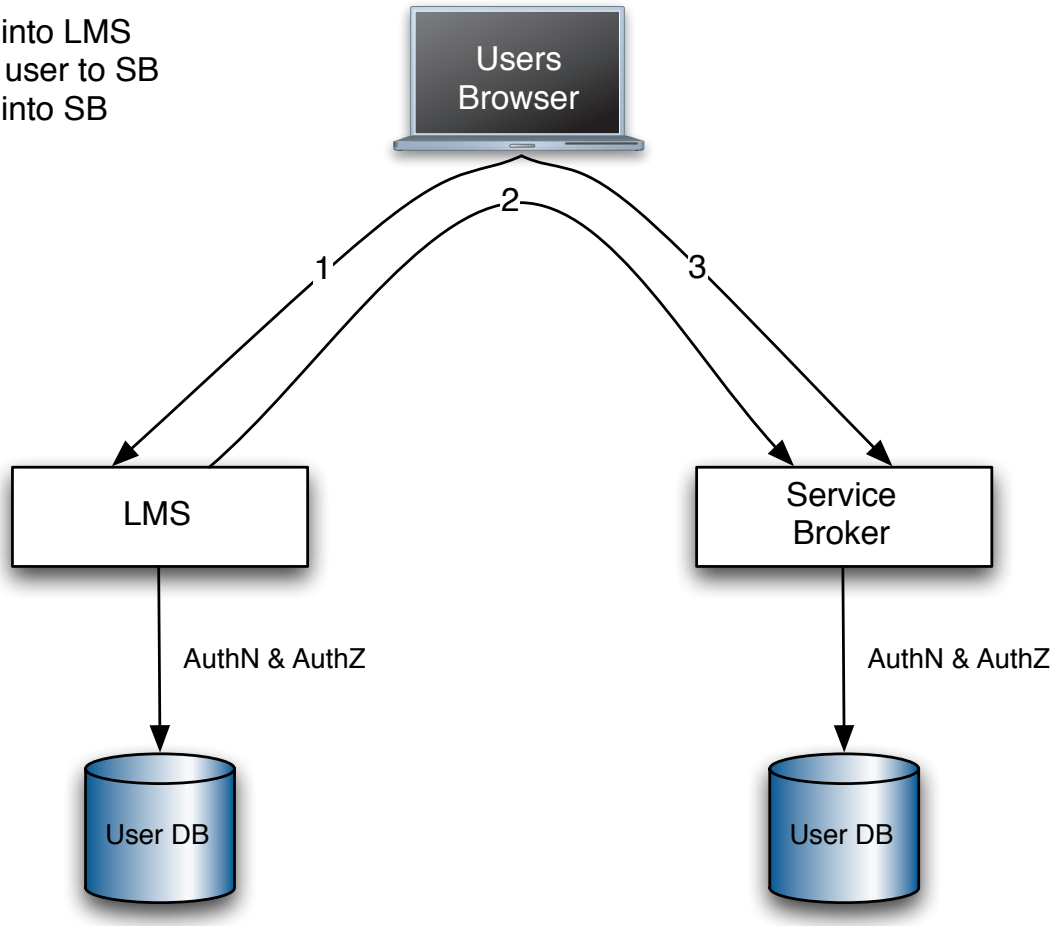

Fig 1. No authentication integration

The iLabs Authentication Proposal is to extend the iLab SB to also support additional AuthN schemes as described below.

SSO Authentication Integration

Many organizations already implement Single-Sign-On (SSO) systems which use an independent, possibly federated, AuthN service which can be integrated into web-based applications such as an LMS through an SSO module or plugin. By integrating such a module into the SB as an AuthN provider, users will be able to be directed from an LMS to the SB without having to perform a separate login. This is illustrated in Fig 2.

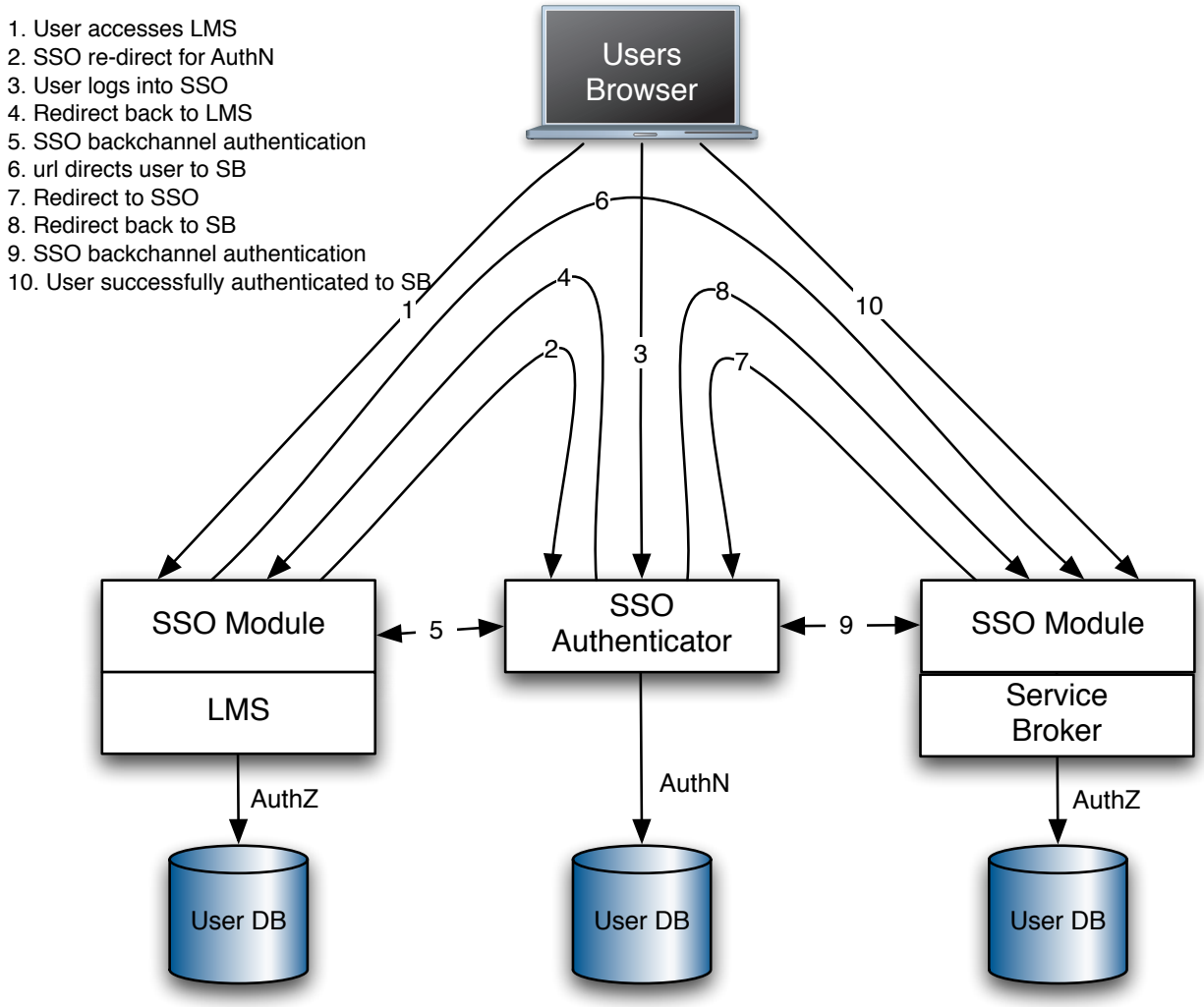

Fig 2. SSO authentication integration

For users who authenticate via the SSO Module, the SB may choose to populate a user's groups with a group list obtained from the SSO system, or solely rely upon group assignments made by the SB's administrator in the SB's User DB.

SSO Modules for common SSOs such as Shibboleth, PubCookie and CAS as well as platform authN services such as LDAP and ActiveDirectory usually integrate with the web server software (e.g. IIS or Apache) and provide a common integration interface to the hosted web application. A benefit of this is that there are unlikely to be any changes to the SB that are specific to any one of these alternatives.

Remote Authentication Integration

Separate to the requirement to support authN by SSO, LDAP or AD there is a requirement to support passing of authN from an external system such as a LMS to the SB. This would enable users of an LMS to lauch directly into experiments from within LMS coursware without additional authN dialogs.

To enable external systems to provide the AuthN to the SB requires the development of a Remote Authentication (RA) module for the external system as well as an RA Web-Service (RA WS) and Remote Authentication capability within the login.aspx for the SB. The user will first login to the LMS and by selecting a link to the RA Module, a call is made to the RA WS at the SB which primes the SB to accept an AuthN performed via query arguments in a redirect url the RA Module sends the users browser to. This interaction is illustrated in Fig 3.

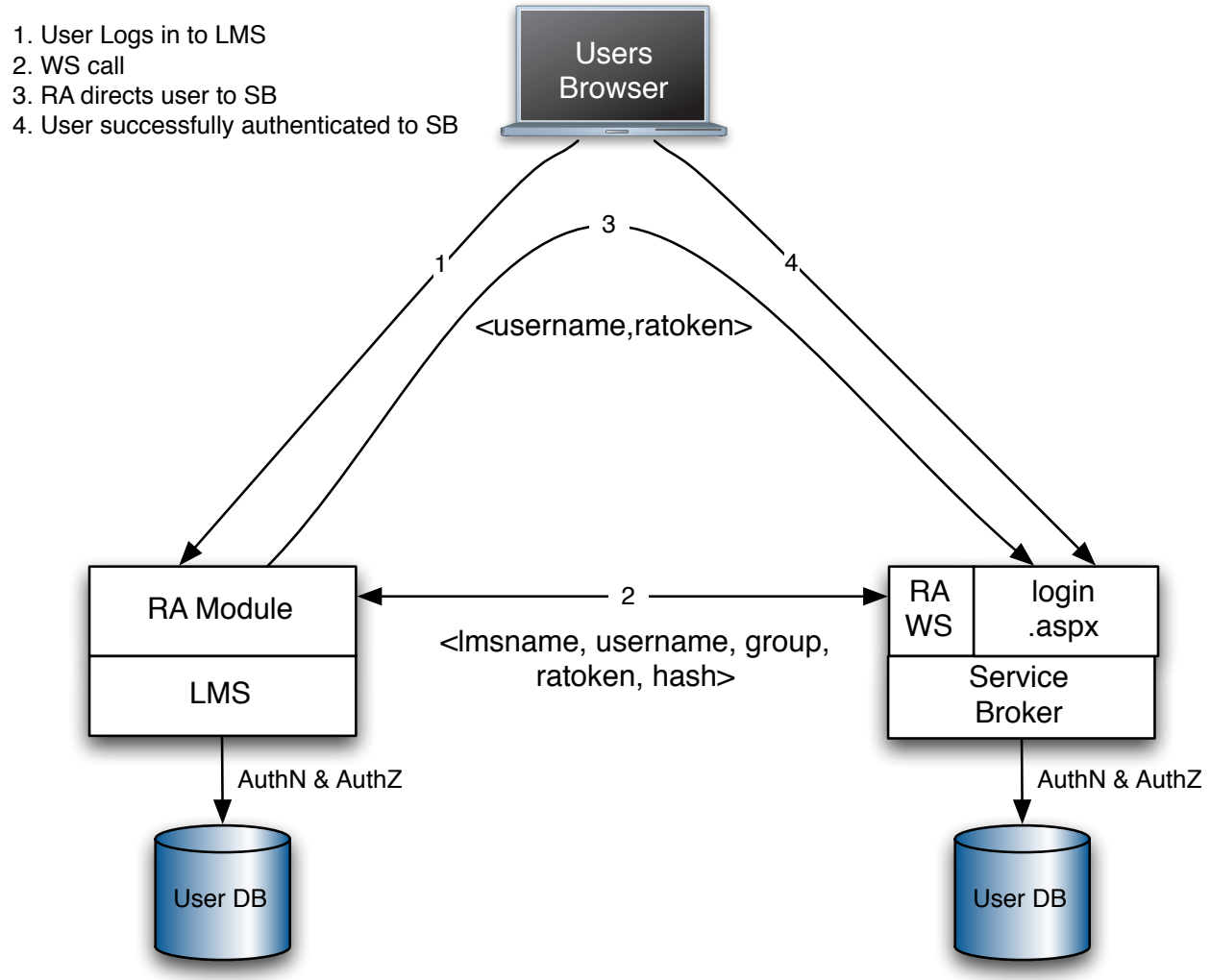

Fig 3. Remote authentication integration

Proposed Implementation
iLabs SB will use ASP.NET Forms Authentication to implement the proposal to support the three AuthN schemes outlined above.

When a users browser accesses a protected url in the SB, as in Fig 4, the session cookie is checked to verify the user is authenticated. If there is no valid session cookie, a redirect to the *login.aspx* defined in the *web.config* file occurs.

Request Resource:
HTTP GET http://sbhost/resource-path?param1=value1&param2=value2&....&paramN=valueN

cookies:<sbsessioncookie=cookiedata, ...>

Redirect to login.aspx:
HTTP GET http://sbhost/login.aspx?target=http%3A%2F%2Fsbhost%2Fresource-path%3Fparam1%3Dvalue1%26param2%3Dvalue2%26....%26paramN%3DvalueN

Fig 4. Resource access from browser

The form presented to the user by *login.aspx* allows the user to select from one or more of the configured methods. A mockup of this form is shown in Fig 5.

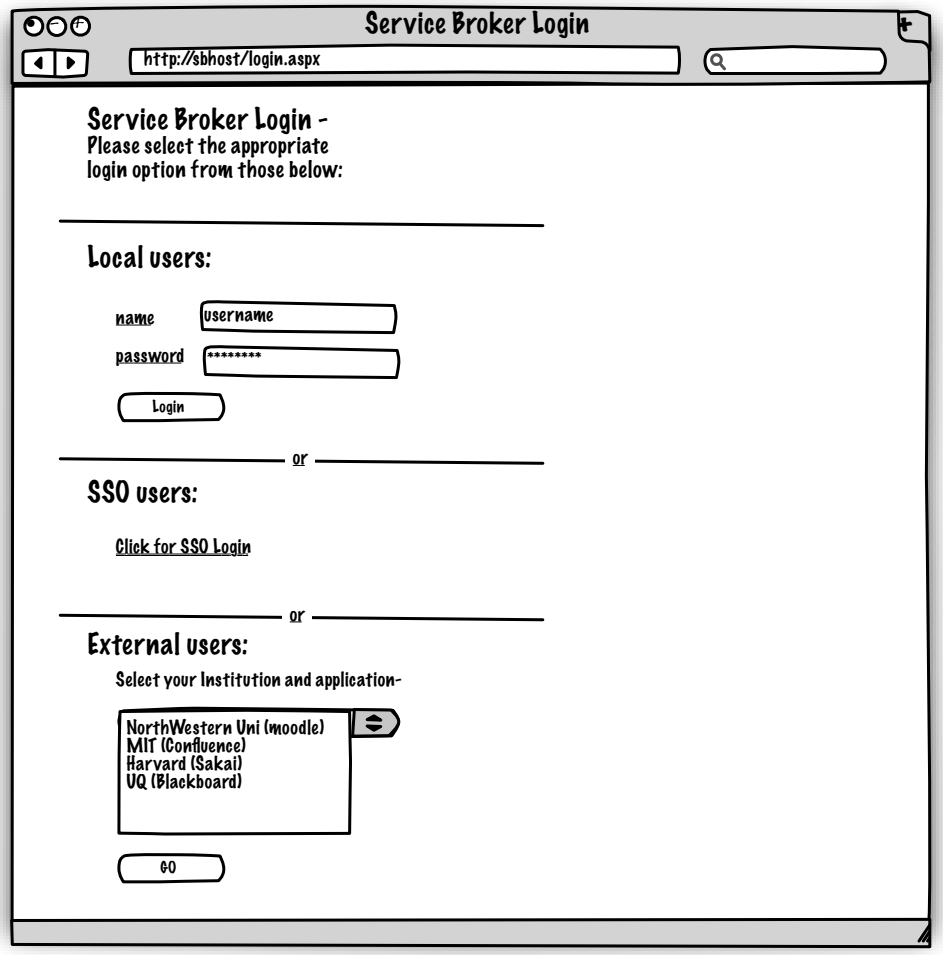

Fig 5. Login Form mockup

The methods include:
1. Local DB account. The username and password are requested via form fields. On submit these are validated against the local database. After successful authentication the *login.aspx* has created a session cookie and the browser is redirected to the originally requested protected url.

2. Single sign-on. The users browser is redirected to the SSO login url. The url of the originally requested SB resource is embedded in the url as a parameter value as shown in Fig 6.

HTTP GET http://sbhost/SSOLoginPath?target=http%3A%2F%2Fsbhost%2Fresource-path%3Fparam1%3Dvalue1%26param2%3Dvalue2%26....%26paramN%3DvalueN

Fig 6. Redirect to SSO login url

After successful authentication the SSO Forms Authenticator invoked at the *SSOLoginPath* has created a session cookie and the browser is redirected to the originally requested protected url.

3. Remote Authentication. To allow users who require remote authentication to access the SB prior to logging into their LMS, the SB *login.aspx* can offer the list of trusted external LMS sites as another authentication selection. This list is constructed from pre-configured data described in the following section. When a user selects one of these, they are redirected to a url at their LMS which invokes the SB remote authentication plugin/module. The url will contain the originally requested resource as a query parameter. Fig 7 shows an example of this redirect url.

HTTP GET http://lmshost/LMSPath/iLabsRAplugin?sb=sbhost&target=http%3A%2F%2Fsbhost%2Fresource-path%3Fparam1%3Dvalue1%26param2%3Dvalue2%26....%26paramN%3DvalueN

Fig 7. Redirect to remote authentication url[1]

If the user does not already have a valid authenticated session cookie with the remote application, they will be forced to authenticate by the remote application. The process from this point is identical to the process described below, where the user begins with a session in the remote application.

[1] Note that this url is equivalent to an embedded url link used in the LMS content to direct a user to a SB resource, as shown below in Fig 8.

Accessing the Service Broker from a remote application using Remote Authentication

The remote authentication method is available for integration with external systems, typically a LMS, which lack a SSO solution.

Pre-configuration establishes a shared secret between each (SB, LMS) pair. Thus each LMS has a table of SB records:

<sbname, sb_ra_ws_url, shared_secret>

Where *sbname* is a unique short name for the SB, *sb_ra_ws_url* is the url of the remote authentication web service at the SB, and *shared_secret* is the preconfigured shared secret for this (SB, LMS) pair.

Similarly the SB has a table of LMS records:

<lmsname, lms_display_name, lms_ra_url, shared_secret, lms_submit_ws_url, group_list>

Where *lmsname* is a unique shortname for the LMS, *lms_display_name* is a user friendly name displayed in the login page pop-down list shown in Fig 5., *lms_ra_url* is the url of the remote authentication plugin at the LMS, *shared_secret* is the preconfigured shared secret for this (SB, LMS) pair, *lms_submit_ws_url is the url for a web service at the LMS used to submit experiment results from the SB* (this service is yet to be defined)*,* and *group_list* is a list of groups this LMS is authorised to use.

Remote Authentication Walkthrough

A user is already authenticated to a trusted external system (LMS etc.) and selects a link, as in Fig 8, to invoke a request to the SB.

Launch the lab1 iLab at service broker sbhost as a member of groupa

Fig 8. Embedded link for remote authentication plugin[2]

The RA module/plugin in the LMS makes a web services call to the SB remote authentication service. This call includes the following data items.

- LMS Name
- User Name
- User Group
- Remote Authentication token (random string)
- possibly other user attributes (email etc.)
- A hash of the above data and the shared secret.

The SB verifies the data by recreating the hash using its copy of the shared secret. If the hashes do not match an error is returned to the LMS and logged locally, no further action is performed at the SB.

[2] Note that this url is equivalent to the redirect url used in the SB login form to direct a user to a LMS for authentication to a SB resource, as shown above in Fig 7.

In the case of a hash match, if the user does not exist in the database, the SB creates the user record with the other attributes added. The SB checks to see if there is a current valid session, if a session exists and the current group is different to the requested group, the current group is changed to the requested group. The password field is set to the Authentication Token and a TTL value is also set to limit validity of the token to a few seconds.
After receiving a successful return of the web service call, the LMS redirects the user to the resource url at the SB. The User Name and Remote Authentication Token are encoded as query parameters in the url as shown in Fig 9.

HTTP GET http://sbhost/resource-path?ilab=lab1&user=jdoe&ratoken=gfyf7665fyf76rfyt6fyy6

Fig 9. Redirect url for Remote Authentication

The SB receiving the request for the resource url will perform the normal session cookie validation and if the cookie is valid the current session attributes, including current group, are used in subsequent authorization decisions.

If the session cookie is not valid, the SB performs a normal redirect to the login.aspx with the resource url encoded as a query parameter in the url. The login.aspx inspects the query parameters prior to presenting the user login form. If the query parameters include a resource url containing user and ratoken parameters, these are validated against the database values for ratoken and ttl value. If valid a session cookie is created, the ttl is set to 0 and the user redirected to the requested resource, as in Fig 10. If invalid an error is shown. If there is no resource url containing user and ratoken parameters, the user login form shown in Fig 5. is displayed.

HTTP GET http://sbhost/resource-path?ilab=lab1

cookies:<sbsessioncookie=cookiedata, ...>

Fig 10. Redirect url after successful Remote Authentication

Future Work

The following items are areas for future development:

- Changes to the authorisation code within iLabs SB.
- Methods for users to submit experiment results back to their LMS system.

Appendix A - Authentication in a Distributed Service Architecture

It has been postulated that as remote labs become more commonly used, that there will be emerging requirements to integrate the services of the Service Broker and Lab Servers with other external services to satisfy scenarios such as workflows involving experiments and publish and subscribe notification services. This appendix attempts to define this environment and a way forward in handling authN/authZ where services use other services without direct involvement of the users web browser, and the session cookies stored there.

A. User Authentication

Currently the Service Broker authenticates users against a local database. The users interact with the service Broker using a web browser over http.

The iLabs Authentication Proposal (this document's main content) adds two new authentication scenarios.

1. The Service Broker will be able to use other authentication services such as Active Directory or Single-Sign-On systems to authenticate users.
2. The Service Broker will be able to accept proof of prior authentication given by appropriately configured external applications such as Learning Management Systems.

In all three authentication scenarios, the maintenance of session validity will be achieved using cookies stored in the users web browser.

B. Service Authentication

The Service Broker and other iLab web services, such as remote Service Brokers, and Lab Servers are all mutually authenticated using pre-shared keys. The Lab Server does not deal with user authentication or authorisation. Essentially, the Lab Server performs experiments anonymously for the Service Broker, and only the Service Broker knows which user is associated with each experiment execution.

C. New distributed service architecture.

This creates difficulty when looking at new functionality where a user or user agent needs to communicate with the Lab Server directly. This direct communication might be for retrieval of status information, or data in long-term experiments, or it might be because a experiment is being run as part of a larger workflow or sequence. We need a method where a user or user agent can provide information to the Lab Server which will enable it to get an assurance from the Service Broker that the user is valid and has been authenticated for the requested service.

A new authentication protocol called OAuth has been designed to allow users who maintain separate accounts with different web services to create mutually authenticated and authorised channels between the web services.

Say a user has accounts with a blogging site and a photo site. The blogging site is able to pull photos from external photo web services, and the photo site is able to provide access to photos via a web service. But the user does not want to make his photos public and doesn't want to store his photo site credentials in his blog site profile. OAuth allows him to

generate a key at his photo site and then configure that key in his Blog site preferences. The key is opaque and provides the blog site with no information about his account at the photo site. But the blog site can use the key in an OAuth protocol exchange with the photo site to gain access to the users photos.

The OAuth capabilities can be translated to the iLabs problem of user agents needing to interact directly with a Lab Server. As described in A. the user can authenticate to the Service Broker using a web browser. Now there is a screen on the Service Broker where the authenticated user can obtain an OAuth key. The user can then store this key in their profile at some new service, such as a workflow engine. The new service, eg. workflow engine, can now provide this key to the Lab Server and the Lab Server will perform an OAuth protocol exchange with the Service Broker. The Service Broker will then provide the Lab Server with confirmation that the user was authenticated and will also be able to provide other services on behalf of the user such as access to experiment data storage. The Lab Server never needs to know who the user is, but is just as assured about the users validity as when the users requests are delivered via the Service Broker.