OAuth for TeraGrid Gateways

Jim Basney <jbasney@ncsa.uiuc.edu>
Jeff Gaynor <jgaynor@ncsa.uiuc.edu>

Table of Contents

Document Status

This document has been reviewed and is now considered stable. Only minor updates are expected from this point. Please send comments to the authors.

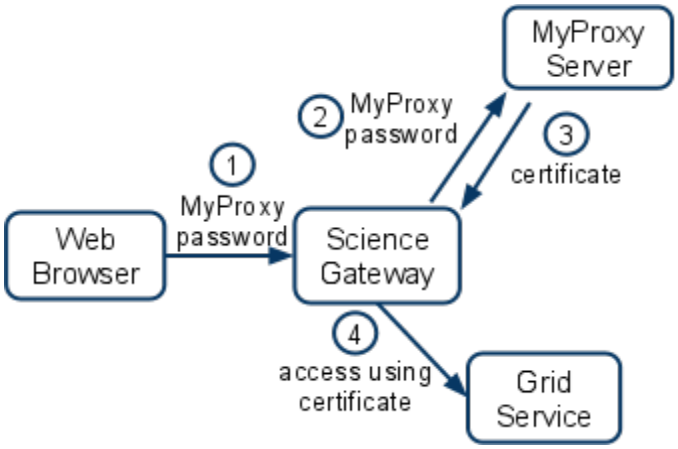
For the latest information about this project, visit: http://security.ncsa.illinois.edu/teragrid-oauth/

Reviews and approvals are summarized in the following table:

Group	Received Comments?	Representative	Approval Date
Gateways	Apr 1 2011	Nancy Wilkins-Diehr	
Globus Online	Mar 21 2011	Rachana Ananthakrishnan	
Operations	Mar 25 2011	Jeff Koerner	
Portals	Mar 25 2011	Maytal Dahan	
RP Forum		John Towns	
Security	Apr 13 2011	Jim Marsteller	Apr 13 2011
User Services		Sergiu Sanielevici	

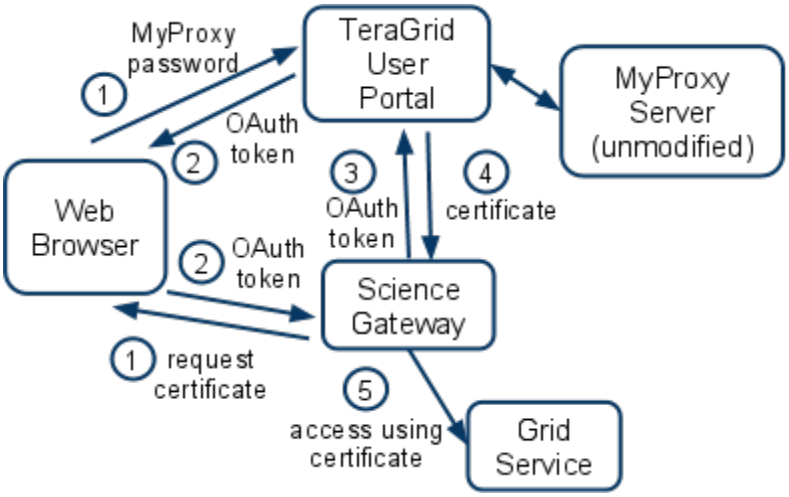
Summary

Currently, when a TeraGrid science gateway allows a researcher to use his or her own TeraGrid account, *rather than using a community account*, for access to TeraGrid resources, the science gateway uses the TeraGrid MyProxy server as illustrated in the following figure:



The researcher provides his or her TeraGrid MyProxy username and password to the science gateway, which the gateway uses to obtain a (short-lived) certificate for the researcher from the TeraGrid MyProxy server, so the gateway can access TeraGrid resources on the researcher's behalf, by authenticating with the researcher's certificate. A significant drawback to this approach is that it discloses the researcher's (typically long-lived) TeraGrid MyProxy password to the science gateway.

To address this drawback, TeraGrid will provide an OAuth service, integrated with the TeraGrid User Portal, as illustrated in the following figure:



When the science gateway requires a certificate to act on the researcher's behalf, it redirects the researcher's browser to the TeraGrid User Portal (TGUP), where the researcher authenticates (as usual) and approves (or denies) use of his or her TeraGrid credentials by the science gateway. If the researcher successfully authenticates and approves the use, the TGUP redirects the researcher's browser back to the science gateway with a one-time-use OAuth token that the gateway uses to obtain a certificate from the TGUP OAuth service, so the gateway can access TeraGrid resources on the researcher's behalf. The TGUP OAuth service will implement the OAuth 1.0a protocol, which is not shown in full detail above, but is specified in RFC 5849. The TGUP OAuth service will require registration of science gateways and only allow authenticated requests from registered gateways.

The result is that TG researchers only enter their TGUP password on the TGUP, rather than also at science gateway web sites.

Note that this approach applies to web browser interactions and does not impact use of *myproxy-logon* on the command-line.

Note also that this approach is not a replacement for the widely adopted community account model for science gateways, which enables gateways to serve users who don't themselves have TeraGrid accounts. *TeraGrid science gateways using the community account model do not need to use this OAuth service.* The OAuth service is only for cases where gateways want to support authentication using individual TeraGrid accounts, as an alternative to the community account model.

Background

In this section we review background material that will inform the design and development of the TGUP OAuth service.

TeraGrid Science Gateways

The TGUP OAuth service supports TeraGrid science gateways. Visit the TeraGrid Science Gateways home page for background materials, and the Science Gateways Use Cases page for details about current gateway implementations.

TGUP InCommon (Shibboleth) Support

The effort to develop and deploy the TGUP OAuth service will be similar in some ways to our previous effort to develop and deploy the TGUP InCommon (Shibboleth) capability. Please refer to the TGUP Shibboleth documentation for design, implementation, and support materials related to that effort. See also our IDtrust paper (and slides) for a description of TeraGrid Single Sign-On (SSO) and how it is integrated with InCommon authentication.

Unvetted and Vetted TGUP Accounts

The design and implementation documents for the TGUP "Unvetted/Vetted" account management process also provide a reference for the TGUP OAuth work, both as an example of how TGUP changes are developed and deployed and as well as providing documentation for TGUP account management processes that the TGUP OAuth service must integrate with.

CILogon

The CILogon Service (https://cilogon.org/) provides an example of using OAuth for certificate issuance to portals. See the CILogon Portal Delegation page for more details, and visit https://demo.cilogon.org for a demonstration. TeraGrid will be modifying existing code from the CILogon project for this work.

OAuth

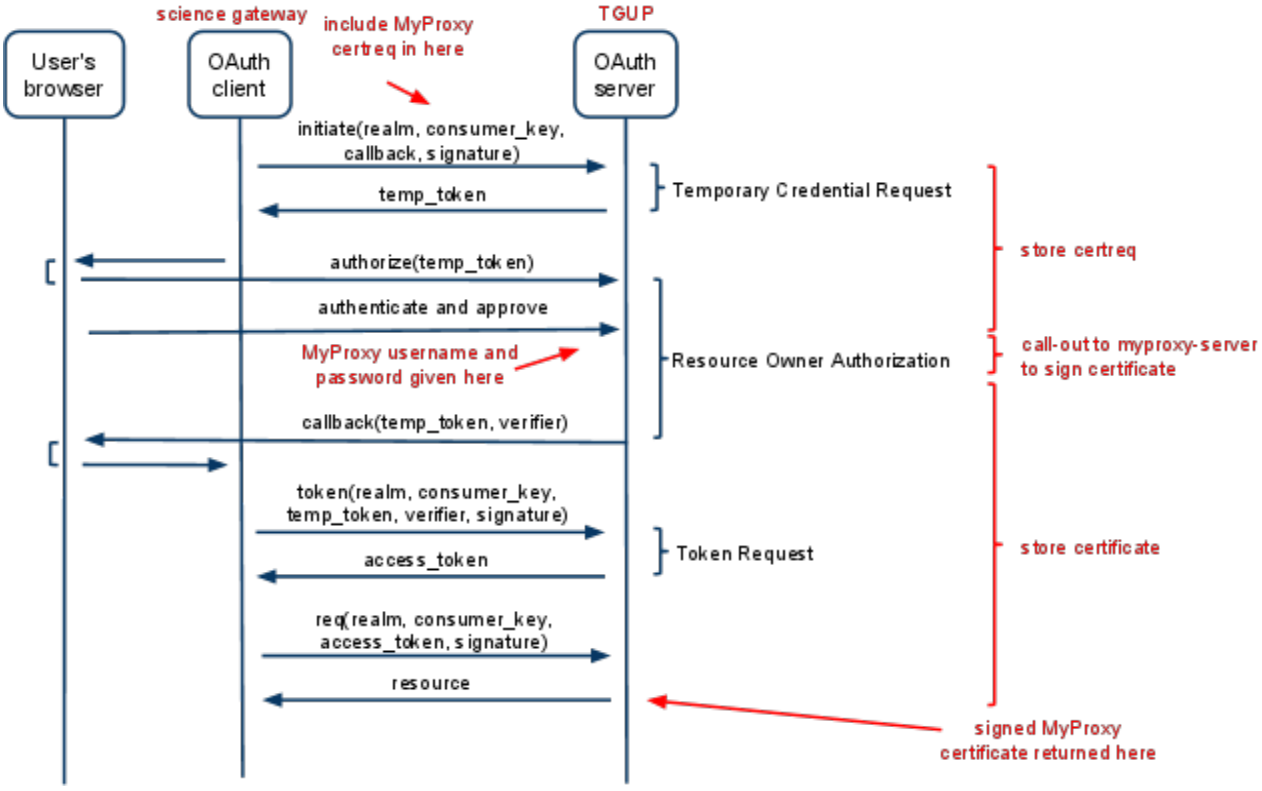
OAuth provides a relatively simple yet powerful mechanism for resource owners (that is, end users) to allow controlled, third-party access to their resources. This requirement is common across many applications, including the science gateways and web portals that are the focus of this work. In OAuth, the resource owner interacts with an OAuth server to approve access to the resource by the third party (the OAuth client) and issue tokens to the third party for secure access to the resource. The use of separate OAuth tokens enables the resource owner to allow access without exposing his or her resource credentials (for example, passwords) to the third party. OAuth tokens can be for one-time use or renewable, can have limited lifetimes, and can limit the scope of access to the resource in various ways. The OAuth 1.0 protocol was initially developed in 2007 and was updated to version 1.0a in April 2009 to address a security flaw. OAuth 1.0a was published as RFC 5849 in April 2010. An update to the protocol, OAuth 2.0, is currently under development in the IETF OAuth working group (see draft-ietf-oauth-v2), and is

expected to be finalized in 2011. Open Source OAuth 1.0 implementations are widely available, and OAuth 1.0 support is included in many Web frameworks.

OAuth is a very flexible security protocol, and its strength depends on how it is used. The management of OAuth keys and secrets and the methods employed for authentication and signing critically impact the overall security of the system. For example, the widely reported compromise of Twitter's use of OAuth in September 2010 was a result of weaknesses in how Twitter employed the OAuth protocol, rather than a vulnerability in OAuth itself (see article). As another example, the question of message signatures versus bearer tokens has been an active topic of discussion in the OAuth 2.0 working group, in part regarding trade-offs between implementation complexity and threat protection in different scenarios (see article and draft-tschofenig-oauth-signature-thoughts). The security profile of an OAuth use case depends on many factors including the type of resource being protected, contacting a well-known service endpoint versus doing dynamic service discovery, and whether the user agent is a web browser, native application, or autonomous service. Our use case here fits a widely adopted OAuth pattern: web browser access to a well-known service endpoint using message signatures.

TGUP OAuth Protocol Overview

The following figure presents the TGUP OAuth protocol flow:



As the figure illustrates, the TGUP OAuth server must support two interfaces: one for interaction with the user's browser, and the other for interaction with the science gateway. In OAuth 1.0a

(RFC 5849) terminology, the TGUP acts as the *OAuth server*, the science gateway acts as the *OAuth client*, and the user (researcher) acts as the *OAuth resource owner*.

The science gateway (OAuth client) initiates the workflow by generating a private key for the user, submitting a certificate request to the OAuth server, and obtaining a "temporary token" from the OAuth server, then redirecting the user's browser (with the "temporary token") to the OAuth server. The user authenticates to the OAuth server (at https://portal.teragrid.org/oauth) using his or her TGUP (MyProxy) username and password and approves the gateway's request for credentials. The OAuth server then redirects the user's browser back to the science gateway (OAuth client) with a "verifier" that the gateway then uses in an OAuth token request to the OAuth server, followed by an OAuth resource request to obtain the stored certificate.

All of these interactions are over HTTPS to provide integrity and confidentiality. All OAuth client messages are signed using the "RSA-SHA1" signature method per RFC 5849. Note that the user's private key is never sent over the network: it is generated locally at the science gateway, and the gateway sends only the public key in a certificate request to the OAuth server for signing by MyProxy.

The full protocol message specification is provided at http://security.ncsa.illinois.edu/teragrid-oauth/.

Design and Implementation

TGUP (OAuth Server)

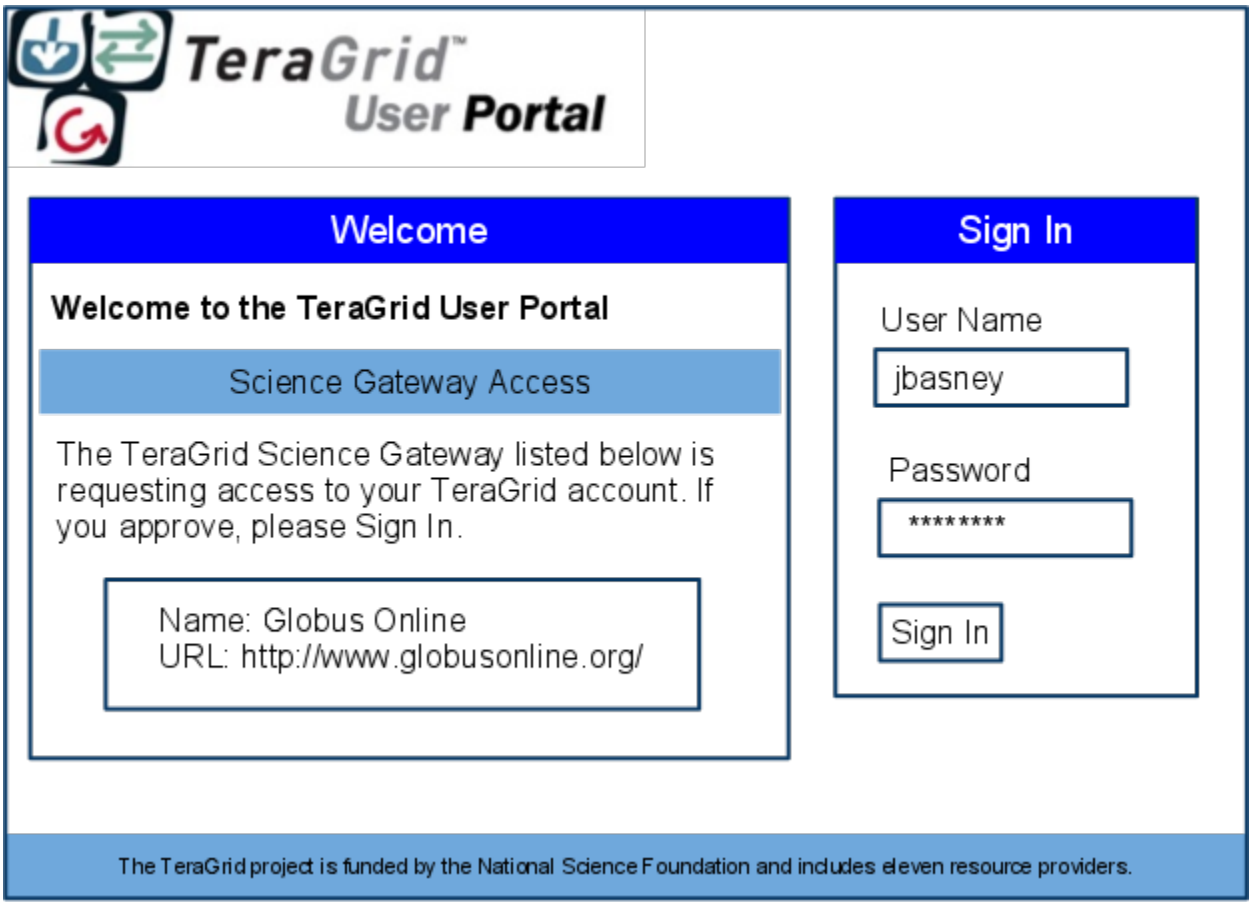
For the TGUP component of the OAuth capability, we will develop an OAuth service (Tomcat Java servlet) that serves requests both from user browsers and from science gateways over HTTPS on port 443. The servlet's browser interface will support customization (CSS) to match the look-and-feel of existing TGUP components. It will run under https://portal.teragrid.org/oauth and will not require changes to the TGUP homepage at https://portal.teragrid.org/.

The OAuth service requires no direct integration with existing TGUP components and has no dependency on Liferay. Instead, it provides its own independent login page and OAuth session management (i.e., there will not be an "OAuth Sign In" option added to the TGUP homepage). However, the OAuth service is co-located with the TGUP to provide a consistent TeraGrid user experience (i.e., users log in at https://portal.teragrid.org/... URLs, whether they're using the TGUP or using OAuth with a science gateway). The servlet runs in the TGUP's Tomcat instance, behind the TGUP's Apache HTTPD in its own URL subdirectory (https://portal.teragrid.org/oauth). The HTTPD is responsible for SSL processing for the HTTPS connections (as with existing TGUP components).

The service has a "whitelist" of gateways (OAuth clients) that it serves, and it provides a web

form for registering new OAuth clients. The service supports fail-over to multiple MyProxy servers (i.e., myproxy.teragrid.org and myproxy.psc.teragrid.org), similar to existing TGUP services.

A mock up of the TGUP OAuth Authorization page (https://portal.teragrid.org/oauth/authorize) follows:



The authorization page serves two important functions: 1) explicitly obtain approval from the TeraGrid user for the gateway's use of the user's account, and 2) prompt the TeraGrid user for his or her TeraGrid username and password. The name and URL for the gateway making the request are read from the OAuth Client Table (see below), according to the information provided by the gateway at registration time. This page will only be shown for digitally signed OAuth requests by approved, registered gateways; in other cases, an error message will be displayed and the TeraGrid user will not be prompted for a password. Upon clicking the "Sign In" button, the TeraGrid user's browser is redirected back to the gateway, so in normal operation, this is the only https://portal.teragrid.org page the user will see during an OAuth transaction.

OAuth Server Data

The TGUP OAuth server is a stateful service. It must store registration records for OAuth clients

(science gateways), which are updated infrequently, and must also store OAuth session data for each protocol transaction in progress. This requires three tables: the OAuth Client Table, the OAuth Client Approval Table, and the OAuth Transaction Table, all of which reside in the same schema, named oauth. We propose to store this schema and all these tables in the local TGUP database, with corresponding users who are granted specific access and with a separate administration account for these tables (see below). For good interactive performance, this will require 0.1s maximum response times for OAuth Transaction Table operations, with a persistent connection pool to the database. The data stored in the (fully qualified) tables:

OAuth Client Table (oauth.clients)

Column name	Type	Comments
oauth_consumer_key+	character varying	The identifier portion of the OAuth client credentials, i.e., a identifier for the gateway.
oauth_client_pubkey	character varying	The OAuth client (gateway) public key.
name	character varying	A name for the OAuth client (gateway) for display to the user.
home_url	character varying	The home page URL for the OAuth client (gateway) for display to the user.
creation_ts	timestamp	A timestamp entry for when the entry was created.
error_url	character varying	A URL for display to the user that the user can visit for assistance in case of errors.
email	character varying	OAuth client (gateway) email address. Used for notification when the client (gateway) is approved.

OAuth Client Approval Table (oauth.client_approvals)

Column name	Type	Comments
oauth_consumer_key+&	character varying	The identifier portion of the OAuth client credentials, i.e., a identifier for the gateway.
approved	boolean	A flag to indicate if this client (gateway) is approved. Set to false when the record is created. OAuth transactions are denied if it is false. Set to true after review by TeraGrid staff.
approval_ts	timestamp	A timestamp entry for when the client was approved.
approver	character	The username of the TG staff person who approved

	varying	the client (gateway).

We store the approved flag in a separate table because it is controlled by a different webapp with different permissions.

OAuth Transaction Table (oauth.transactions)

Column name	Type	Comments
temp_token+	character varying	The OAuth temporary credentials identifier, including a timestamp for enforcing a limited lifetime on transactions. This is the oauth_token returned in the temporary credential request.
temp_token_valid	boolean	A flag to indicate if the temp_token_secret is valid (i.e., has not yet been used). Once the temp_token_secret is used, it is marked invalid and can not be used again.
oauth_callback	character varying	An absolute URL back to which the server will redirect the resource owner when the resource owner authorization step is completed.
certreq	character varying	The certificate request submitted by the OAuth client (gateway) in the temporary credential request.
oauth_consumer_key&	character varying	The OAuth client identifier, i.e., the primary key for the associated OAuth Client Table record.
oauth_verifier	character varying	The OAuth verification code sent from the OAuth server to the OAuth client via the user agent during resource owner authorization.
access_token*	character varying	The OAuth access token identifier. This is the oauth_token returned in the OAuth token request.
access_token_valid	boolean	A flag to indicate if the access_token_secret is valid (i.e., has not yet been used). Once the access_token_secret is used, it is marked invalid and can not be used again.
certificate	character varying	The certificate issued by MyProxy to be returned to the OAuth client (gateway) by the OAuth server when a valid access token is presented.

+ = primary key
& = foreign key
* = indexed

Access rights

The specific users for these tables are tg_client, tg_approver, tg_portal, *i.e.*, these are the database users used by the various applications. Access rights are summarized in the chart below (schema omitted):

	clients	client_approvals	transactions
tgup_client	RW	R	-
tgup_approver	R	RW	-
tgup_portal	R	R	RW

where

R = Read access
W = Write access

For instance (detailed below), the science gateway submits a request for registration. A TG staff person runs an application which accesses the database as user “tgup_client”.

Science Gateways (OAuth Client)

For the science gateway component of the OAuth capability, we will provide client libraries that science gateway developers can integrate with. Since we use the standard OAuth 1.0a protocol, we can use existing OAuth client implementations. We will provide Java and Python client libraries.

As with the CILogon OAuth Client API, the TeraGrid OAuth Client API will provide a very simple interface for science gateways. The gateway first calls a requestCertificate() function, that initiates the transaction with the TGUP OAuth server and returns 1) the newly generated private key associated with the certificate request, and 2) a URL for redirecting the user’s browser to the TGUP OAuth service. The gateway then redirects the user’s browser to the provided URL, where the user authenticates at the TGUP OAuth service, and then the TGUP OAuth service redirects the user back to the gateway’s registered callback URL. When the user returns to the gateway, the gateway makes a second API call, to the getCertificate() function, passing the oauth_token and oauth_verifier values from the callback as input to the function. The getCertificate() function completes the transaction with the TGUP OAuth server and returns the X.509 certificate for the user. As described above, the API handles generation of private keys and certificate requests for the gateway. The specification for the Java and Python API calls is provided in a later section below.

Each science gateway must complete the OAuth client registration process via a TGUP web form and be approved for TeraGrid OAuth access before using the TeraGrid OAuth service. The registration process is:

- A science gateway operator submits the OAuth registration form providing the following information:
    - **Science Gateway Name**: A friendly name for the science gateway for display to users on the TGUP OAuth Authorization page (see above).
    - **Science Gateway Home URL**: The home page URL for the science gateway for display to users on the TGUP OAuth Authorization page.
    - **Science Gateway URL**: A gateway-specific URL for display to the user in case of problems that tells the user how to get assistance from the gateway operators.
    - **Science Gateway Email Address**: An email address for notifying the science gateway operator when the gateway is approved for OAuth client access and when any other operational notices are needed.
    - **Science Gateway RSA Public Key**: A 2048 bit RSA public key corresponding to the RSA private key that the science gateway will use to sign OAuth messages.
- The form output includes the oauth_consumer_key (OAuth client identifier) generated by the OAuth server for this new OAuth client. The science gateway operator will need to configure the OAuth client software to use this oauth_consumer_key value when interacting with the TG OAuth server.
- TG staff (Science Gateways Area Director) are notified and review the submission.
- If the submission is approved, a TG staff person runs an application to update the "approved" flag in the OAuth Client Table (see below) and send a notification to the registered science gateway email address.
- The science gateway can now submit OAuth transactions.

Gateway operators can use the following commands to generate their RSA public/private keys for use with OAuth:

```
openssl genrsa -out oauth-privkey.pem 2048
chmod 0600 oauth-privkey.pem
openssl rsa -in oauth-privkey.pem -pubout -out oauth-pubkey.pem
```

They then configure the OAuth client software to use the `oauth-privkey.pem` file (and keep that file private/secure) and submit the contents of the `oauth-pubkey.pem` file in the OAuth registration form.

MyProxy

We do not expect to require any changes to the TeraGrid MyProxy servers (myproxy.teragrid.org and myproxy.psc.teragrid.org) to support this service. The MyProxy servers will continue to accept TeraGrid password authentication from the TGUP for issuance of certificates with lifetimes up to 11 days (264 hours).

Project Plan

1. [Target: March 2011] Design Complete. Sign-offs obtained on this document.
2. [Target: May 2011] Software Complete. OAuth client libraries (Java & Python) and service (Java) implemented (by NCSA).
3. [Target: July 2011] TGUP Integration Complete.
    a. Acceptance Test with Globus Online (using staging/test TGUP instance).
    b. Consistent user interface achieved.
    c. Gateway approval app deployed (developed by TGUP team).
    d. Roll-out to production. Official announcement to gateways.

Design Decisions

The design of the TeraGrid OAuth service was informed by discussions with the TGUP developers as well as the TeraGrid gateways and security working groups. Key design decisions were:

- **No changes to the TeraGrid Liferay portal**: Developing the OAuth service as a web application separate from the TeraGrid Liferay portal avoided unnecessary dependencies for development and deployment of the service. The software developers did not need to become Liferay experts, and the TGUP team can deploy the OAuth service on the servers that are supporting the Liferay service or on different servers (all behind the portal.teragrid.org load-balancer) as appropriate to the TGUP production environment.
- **No changes to TeraGrid MyProxy servers**: The TeraGrid MyProxy servers operate as certification authorities and must be highly secured with strict security policies. MyProxy server changes have implications on certificate policies and TeraGrid operational security. Using the existing MyProxy password-based GET method, already used by the TGUP, avoids MyProxy server changes. We designed the protocol (as described in Section 3) so the OAuth service could issue the MyProxy GET request immediately when the user enters his or her password (i.e., requiring the gateway to provide a certificate request earlier in the protocol), so the user's password is never stored and the MyProxy server knows the user authorized the request by providing the password.
- **Support database persistence for replication**: Storing server state in a replicated database as described in Section 2.2 enables load balancing, fail-over, and replication of the OAuth Service consistent with current TGUP operations.
- **Initially support only password-based authentication**: To simplify our initial design, we focused on password-based authentication only, leaving support for federated authentication [4] as a future enhancement.
- **No support for certificate renewal**: Per TeraGrid policies, the MyProxy servers issue certificates valid for a maximum of 11 days. Allowing gateways to get another certificate for the user when the current user certificate nears expiration without requiring

user intervention would provide better support for long-running jobs. However, it adds complexity and increased risk, so for our initial version, we require explicit user authentication and approval for every certificate issuance.

- **Require authentication and approval for every certificate issuance**: Requiring explicit user authentication and approval for each request addresses security concerns (see Section 5) and simplifies MyProxy integration (because the password is provided in every GET request). We expect approval of each request will not be too inconvenient for TeraGrid users, who typically use only one or two gateways targeted to their area of science, so users would need to approve just once or twice a week (for issuance of certificates valid for up to 11 days). This decision also avoids a motivation for integration with the TeraGrid Liferay portal, since there is no benefit for sharing authentication sessions between Liferay and OAuth. Additionally, it eliminates the need for a separate interface for revoking approvals, as seen in other OAuth services, because all OAuth approvals are one-time only.

RFC 5849 Security Considerations

RFC 5849 identifies 15 security considerations for the OAuth protocol. In this section, we discuss each security consideration in the order it appears in RFC 5849:

1. **RSA-SHA1 Signature Method**: The TeraGrid OAuth service uses the RSA-SHA1 signature method, so it relies “on the secrecy of the private key used by the client to sign requests” (quoting RFC 5849). In case this private key is suspected or known to be compromised, the corresponding public key must be removed promptly from the OAuth Client Table so the OAuth service no longer accepts signatures created by this key.
2. **Confidentiality of Requests**: The use of HTTPS for all network connections provides confidentiality of requests.
3. **Spoofing by Counterfeit Servers**: The use of HTTPS for all network connections provides for server authentication.
4. **Proxying and Caching of Authenticated Content**: The use of HTTPS for all network connections avoids caching of sensitive message content by HTTP proxies.
5. **Plaintext Storage of Credentials**: The use of the RSA-SHA1 signature method (in contrast to other OAuth signature methods) avoids plaintext storage of long-lived secrets on the OAuth server, thereby eliminating this risk.
6. **Secrecy of the Client Credentials**: OAuth client credentials are controlled by trusted gateway operators, unlike other uses of OAuth where the client may be operated by untrusted parties. While RFC 5849 recommends use of additional client authentication factors, such as IP address, we feel that maintaining a registry of gateway IP addresses is too onerous. Instead, we mitigate this risk through active monitoring of OAuth client transactions to detect credential misuse.
7. **Phishing Attacks**: The TeraGrid OAuth service enables us to train TeraGrid users to enter their TeraGrid passwords only on the https://portal.teragrid.org web site, to better protect against phishing attacks where rogue sites ask for TeraGrid passwords. TeraGrid users can verify the portal.teragrid.org URL in their browser and confirm the

authenticated HTTPS connection prior to entering their TeraGrid password.

8. **Scoping of Access Requests**: The current TeraGrid security infrastructure does not support fine-grained delegation of access rights, so it is not currently possible to scope the access rights granted by the user to the science gateway, apart from limiting the lifetime of the certificate issued to the science gateway to 11 days (264 hours) or less. This is a risk that must currently be mitigated by other means (for example, by active monitoring of TeraGrid accesses).
9. **Entropy of Secrets**: The use of HTTPS for all network connections protects against eavesdroppers obtaining secrets for performing brute-force attacks. Additionally, we take care to generate cryptographically strong tokens and verifiers in the OAuth server and enforce a short (15 minute) validity period on the use of these secrets.
10. **Denial of Service / Resource-Exhaustion Attacks**: Accepting requests only from registered OAuth clients offers some protection against DoS attacks. To address resource-exhaustion attacks (for tracking of nonces, tokens, and verifiers for pending transactions), the OAuth server requires transactions to complete within a short period (15 minutes) and rejects requests with timestamps outside that time period. Therefore, OAuth clients must maintain accurate system clocks to include accurate timestamps in OAuth requests.
11. **SHA-1 Cryptographic Attacks**: The TeraGrid OAuth service currently supports SHA-1 signatures for compliance with RFC 5849, but we understand that SHA-1 has known cryptographic weaknesses against collision attacks. We expect to address this in the future by migrating to OAuth 2.0 when it is standardized.
12. **Signature Base String Limitations**: The use of HTTPS for all network connections provides full message integrity, thereby addressing limitations in the OAuth signature method identified by RFC 5849.
13. **Cross-Site Request Forgery (CSRF)**: The TeraGrid OAuth service protects against CSRF attacks by requiring user password entry prior to any token issuance and not relying on cookies or other session state for authenticating browsers.
14. **User Interface Redress**: The TeraGrid OAuth service protects against user interface redress (“clickjacking”) attacks on the authorization page by using an X-Frame-Options: deny HTTP header and by requiring user password entry prior to any token issuance.
15. **Automatic Processing of Repeat Authorizations**: Unlike other OAuth services, the TeraGrid OAuth service does not support automatic processing of repeat authorizations, where the server would grant access based on prior approval, and therefore is not subject to the risks associated with this method. The TeraGrid OAuth service requires explicit user authentication and approval for each request and invalidates OAuth tokens upon use, so they can be used only once.

Other Security Considerations

1. All interactions with the OAuth service are over HTTPS (port 443), over an encrypted TLS channel with server authentication (using a portal.teragrid.org HTTPS certificate).
2. Science gateways generate private keys locally and never send them over the network.

A science gateway sends a certificate request to the TGUP OAuth service to obtain a signed certificate. The TGUP never sees the user private keys.

3. TeraGrid MyProxy servers issue certificates valid for up to 11 days (264 hours), consistent with the EUGridPMA Guidelines on Private Key Protection and NCSA CA Policies.
4. The gateway user enters his or her TGUP password via a web browser only on the TGUP login page (i.e., with a consistent TGUP look-and-feel and an https://portal.teragrid.org URL) over an encrypted TLS channel with server authentication. The TGUP sends the password for verification to the TeraGrid MyProxy server over an encrypted TLS channel with server authentication. Science gateways never see TGUP passwords.
5. The gateway user approves each certificate issuance by clicking a button on the TGUP.
6. OAuth clients (science gateways) must protect their credentials (i.e., private key associated with oauth_client_pubkey) (see RFC 5849 Section 4.6). If compromise of the OAuth client credentials is detected, access may be revoked by setting the approved flag to false in the OAuth Client Table.
7. The TGUP OAuth service uses the RSA-SHA1 signature method specified in RFC 5849 Section 3.4. This method avoids the plaintext storage of credentials by the OAuth server (see RFC 5849 Section 4.5), by using asymmetric cryptography (needing only a public key to be stored) instead of symmetric cryptography (shared secret).
8. See also the Security Considerations section of RFC 5849 (OAuth 1.0a Protocol).
9. The OAuth service logs all successful and unsuccessful authentication attempts using log4j to the TGUP Tomcat logs. Log entries include: timestamp, browser IP address, TG username, OAuth client (gateway) identifier, and OAuth client (gateway) IP address.
10. In case of reported compromise of a TeraGrid password, current standard practice includes revocation of any currently valid certificates issued for the compromised TeraGrid account by TeraGrid MyProxy servers.

Operational Considerations

1. The TGUP OAuth service depends on the TeraGrid MyProxy service, which is replicated at NCSA (myproxy.teragrid.org) and PSC (myproxy.psc.teragrid.org). The TeraGrid MyProxy service depends on the TeraGrid Kerberos service (for verifying TeraGrid usernames and passwords), which is replicated at NCSA and PSC.
2. The TGUP OAuth service is replicated according to the current TGUP replication method (one production instance, two warm backups, load balancer in front). It runs under the same Tomcat instance as Liferay.
3. A workflow is required for registration and approval of OAuth clients (gateways). Who should be notified of OAuth client registration requests (likely some combination of TG security, TGUP staff, and TG gateway coordinator)? Who should review and approve the requests? Who should have the ability to undo approvals (i.e., likely the security-wg will need this capability for incident response) to disable access by an OAuth client?
4. The TGUP OAuth service will be monitored by Nagios and Inca by doing periodic

HTTPS GETs to https://portal.teragrid.org/oauth/initiate and https://portal.teragrid.org/oauth/register.

Issues/Questions

1. Does the TGUP OAuth service support InCommon authentication or only TGUP password authentication? Supporting only TGUP password authentication significantly simplifies the design. **Decision**: Support only TGUP password authentication for now, until we gain more experience with TGUP InCommon authentication.
2. Will the TGUP OAuth service run on the "secure server" or on the regular TGUP instance(s)? No motivation for running on the TGUP "secure server" has yet been identified. The TGUP OAuth service behaves very simlarly to the Liferay instance(s), i.e., accepts TGUP passwords and obtains certificates from MyProxy. **Decision**: Run on the regular TGUP instances (primary and backups).
3. What data store will the TGUP OAuth service use? The TGCDB was considered but eliminated as a candidate because of performance concerns. Should the same data store be used for long-term data (OAuth Client Table) as well as session data (OAuth Transaction Table)? Should session data be stored only in memory (i.e., will the OAuth service be replicated)? **Decision**: Store all data in the TGUP MySQL database to facilitate replication / fail-over.
4. Should sessions be shared between the OAuth service and Liferay? In other words, if a user logs in at the OAuth service, should that also log them in to Liferay (and vice versa)? This would be complicated to implement and is likely not worth the effort or complexity. **Decision**: Avoid all dependencies on Liferay, so do not share session information between OAuth and Liferay. The TGUP Sign In and TG OAuth Sign In pages will be clearly differentiated and will not reference each other or link to each other, so as to minimize user confusion.
5. Is the OAuth client whitelist required to restrict access to only approved TeraGrid Science Gateways? If yes, who will be responsible for approving additions to the whitelist? **Decision**: Require gateway registration and explicit approval by the Science Gateways Area Director (i.e., Nancy).
6. Is the OAuth service replicated? This has implications for data storage (Issue #3). **Decision**: Support replication consistent with the current TGUP deployment, using the TGUP DB as shared storage for all OAuth state.
7. Should OAuth client (science gateway) IP addresses be registered and verified? RFC 5849 Section 4.6 recommends registering OAuth client IP addresses and using IP addresses in addition to OAuth client credentials to verify client identities. This could add a significant management burden for the science gateways and TeraGrid staff to maintain, however. **Decision**: Address this issue by logging all access (with IP addresses in log messages) and monitoring access logs rather than requiring strict IP-based access control.
8. Which signature method should the OAuth service use? RFC 5849 specifies three signature methods: HMAC-SHA1, RSA-SHA1, and PLAINTEXT. Both HMAC-SHA1 and

PLAINTEXT are “shared secret” methods, meaning that they require a cleartext secret to be shared (and stored) by the OAuth client and server, bringing clear security risks (see RFC 5849 Section 4.5). The RSA-SHA1 method uses asymmetric cryptography, so the OAuth server need only store the client’s public key to verify signatures, rather than generating and storing shared secrets. The likely drawbacks to the RSA-SHA1 method are: 1) asymmetric cryptography is slower and 2) generating and managing asymmetric keys can be more complex for clients. **Decision**: Use RSA-SHA1 to avoid the risks and complexity on the server-side of storing and generating shared secrets.

9. Should we have a webapp that lets users manage (view and revoke) their OAuth approvals? This is common for OAuth services that support long-lived approvals (e.g., connecting Facebook and Twitter accounts). In the case of the TGUP OAuth service, the user must explicity sign in and approve each certificate issuance to a gateway (i.e., there is no “remember” checkbox). **Decision**: Because we require explicit user approval every time, there are no long-lived approvals to be viewed and revoked, so no webapp is needed for this.
10. Should we support certificate renewal, i.e., allowing gateways to get another certificate for the user when the current user certificate nears expiration, without requiring user intervention? The TG OAuth service will issue certificates valid for up to 11 days (264 hours) to gateways. For long-running jobs, if some renewal facility is not provided, gateway users will need to sign in every 11 days so the gateway has a valid certificate for them. **Decision**: Certificate renewal adds significantly complexity, so we will not support it in the initial TG OAuth service.

API specification

This section gives the exact syntax for the two API calls required by the Science Gateways. The currently supported languages are Java and Python.

Java

This consists of a TeraGridOAuthService service class with precisely two methods.

```
public Response requestCert();
```

- Action: A private key and corresponding certificate request are generated.
- Result: A response object containing the redirect URI for the client browser and a PrivateKey object.
- This depends on config parameters (OAuth client key, OAuth server URL, etc.) specified in either the service’s web.xml file or in an XML config file

```
public X509Certificate getCert(String tempToken, String verifier);
```

- `tempToken` which is returned by the service as part of the callback.
- `verifier` which is returned by the service as part of the callback.
- Action: The rest of the delegation action is performed and the certificate is acquired.

- Result: The returned certificate.

Python

This consists of a TGUP service class with precisely two methods.

`requestCert()`

- Action: A private key and corresponding certificate request are generated.
- Result: The private key and the redirect URI for the client browser are returned. The URI includes the temporary token which should be stored in the user's session state for future use.
- This depends on configuration information (client private key, name and OAuth service URI) specified either in the deployment or in a separate XML configuration file.

`getCert(tempToken, verifier)`

- **`tempToken`** which is returned by the service as part of the callback.
- **`verifier`** which is returned by the service as part of the callback.
- Action: The rest of the delegation action is performed and the certificate is acquired.
- Result: A byte array containing the DER-encoded certificate.