An OAuth Service for Issuing Certificates to Science Gateways for TeraGrid Users

Jim Basney and Jeff Gaynor
National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign
1205 West Clark Street, Urbana, Illinois 61801
{jbasney,gaynor}@illinois.edu

ABSTRACT

In this paper, we present a TeraGrid OAuth service, integrated with the TeraGrid User Portal and TeraGrid MyProxy service, that provides certificates to science gateways. The OAuth service eliminates the need for TeraGrid users to disclose their TeraGrid passwords to science gateways when accessing their individual TeraGrid accounts via gateway interfaces. Instead, TeraGrid users authenticate at the TeraGrid User Portal to approve issuance of a certificate by MyProxy to the science gateway they are using. We present the design and implementation of the TeraGrid OAuth service, describe the underlying network protocol, and discuss design decisions and security considerations we made while developing the service in consultation with TeraGrid working groups and staff.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection – *authentication.*

General Terms

Security

Keywords

Science Gateways, TeraGrid, OAuth, PKI, MyProxy

1. INTRODUCTION

TeraGrid science gateways [9][10][11] provide access to high-end resources through community designed and supported web interfaces. Gateways support both current TeraGrid users and users who are new to TeraGrid. Gateway users may not even know that they are using TeraGrid resources as part of their work. By using TeraGrid *community accounts*, gateways can scale access to a large number of users, without requiring those users to register for their own TeraGrid accounts [5][13].

However, for those gateway users who have their own TeraGrid accounts, it is preferable to use their individual TeraGrid account with the gateway, rather than the gateway's community account, for more accurate accounting and broader access to TeraGrid resources. In this paper we present a mechanism to allow these users to access gateways via their individual TeraGrid accounts in a uniform and secure manner.

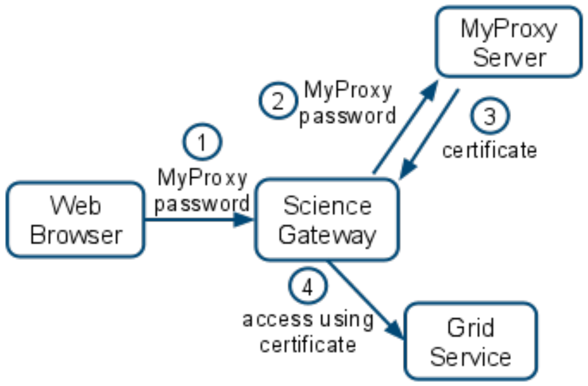


Figure 1. MyProxy Use by Science Gateways Today

Current Approach: Currently, when a TeraGrid science gateway allows access to TeraGrid resources via an individual account rather than a community account, the gateway uses the TeraGrid MyProxy server [3] as illustrated in Figure 1.

The user provides his or her TeraGrid MyProxy username and password to the science gateway (Step 1), which the gateway uses to obtain a (short-lived) certificate for the user from a TeraGrid MyProxy server (Steps 2-3), so the gateway can access TeraGrid resources on the user's behalf, by authenticating with the user's certificate (Step 4). A significant drawback to this approach is that it discloses the user's (typically long-lived) TeraGrid MyProxy password to the science gateway. Many TeraGrid gateways are not operated by TeraGrid staff on TeraGrid networks and do not have a teragrid.org address. Therefore, disclosure of TeraGrid passwords to these systems introduces increased risk to TeraGrid.

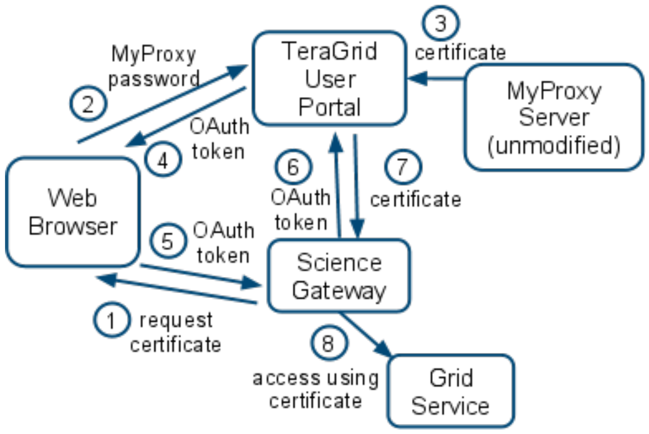


Figure 2. Science Gateways Using OAuth with MyProxy

New Approach: To address this drawback, we have developed a TeraGrid OAuth service, integrated with the TeraGrid User Portal (TGUP), as illustrated in Figure 2. In this new approach, users provide their TeraGrid MyProxy password only to the TGUP, a trusted service operated by TeraGrid staff on TeraGrid networks, so TeraGrid passwords are not disclosed to non-TeraGrid systems.

When the science gateway requires a certificate to act on a user's behalf, it redirects the user's browser to the TeraGrid User Portal (Step 1), where the user authenticates (as with a regular TGUP session) and approves or denies use of his or her TeraGrid credentials by the science gateway (Step 2). If the user successfully authenticates and approves the use, the TGUP obtains a certificate from the TeraGrid MyProxy server (Step 3), then redirects the user's browser back to the science gateway with a one-time-use OAuth token (Steps 4-5) that the gateway uses to get the certificate from the TeraGrid OAuth service (Steps 6-7), so the gateway can access TeraGrid resources on the user's behalf (Step 7). The service implements the OAuth 1.0a protocol as specified in RFC 5849 [1]. To ensure credentials are provided only to approved gateways, the service requires gateways to register prior to using the service.

The remainder of our paper is organized as follows. In Section 2 we present the design and implementation of the TeraGrid OAuth Service. In Section 3 we present the underlying network protocol. Then we discuss design decisions (Section 4) and security considerations (Section 5) that we made while developing the service in consultation with TeraGrid working groups and staff. Lastly, we discuss related work (Section 6) and future work (Section 7), and then we present our conclusions (Section 8).

2. DESIGN AND IMPLEMENTATION

In this section we discuss the design and implementation of the OAuth service from both the server and client perspective.

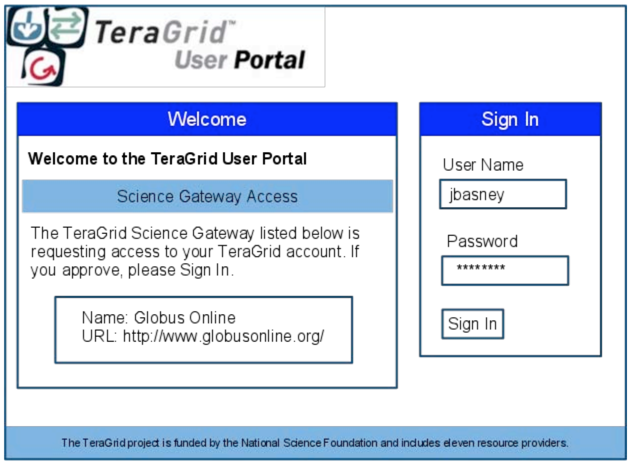
2.1 OAuth Server

The OAuth server is a Java servlet that serves requests from user browsers and science gateways over HTTPS [2] on port 443. The servlet's browser interface supports customization (via CSS) to match the look-and-feel of existing TGUP components. It runs at https://portal.teragrid.org/oauth and does not necessitate any 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 the TGUP's use of the Liferay portal software. Instead, it provides its own independent login page and OAuth session management (i.e., there is no "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 are 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 space (https://portal.teragrid.org/oauth). The HTTPD is responsible for TLS [12] processing for the HTTPS connections (as with existing TGUP components).

The service has a "whitelist" of gateways (OAuth clients) that it serves, and the service provides a web page for registering new OAuth clients. The service supports fail-over to the multiple TeraGrid MyProxy server instances, similar to existing TGUP services.

Figure shows a design mock up of the TGUP OAuth Authorization page (https://portal.teragrid.org/oauth/authorize). The authorization page serves two important functions: 1) explicitly obtain approval from the TeraGrid user for the gateway's use of the user's individual TeraGrid 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 is shown only for digitally signed OAuth requests made by approved, registered gateways; in other cases, an error message is displayed and the TeraGrid user is not 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.



Welcome to the TeraGrid User Portal

Science Gateway Access

The TeraGrid Science Gateway listed below is requesting access to your TeraGrid account. If you approve, please Sign In.

Name: Globus Online
URL: http://www.globusonline.org/

User Name: jbasney

Password: ********

Sign In

The TeraGrid project is funded by the National Science Foundation and includes eleven resource providers.

Figure 3. OAuth Authorization Page

2.2 OAuth Server State

The OAuth server is a stateful service. It must store registration records for OAuth clients (science gateways) and must also store OAuth session data for each protocol transaction in progress. This requires three database tables: the OAuth Client Table, the OAuth Client Approval Table, and the OAuth Transaction Table.

The OAuth Client Table contains the following rows:

- **oauth_consumer_key**: The identifier portion of the OAuth client credentials, i.e., an identifier for the gateway, assigned by the OAuth server at registration time.
- **oauth_client_pubkey**: The OAuth client (gateway) public key, generated by the client and registered with the server.
- **name**: A name for the OAuth client (gateway) for display to the user, provided by the client at registration time.
- **home_url**: The home page URL for the OAuth client (gateway) for display to the user, provided by the client at registration time.
- **creation_ts**: A timestamp entry for when the entry was created.
- **error_url**: A URL for display to the user that the user can visit for assistance in case of errors, provided by the client at registration time.
- **email**: OAuth client (gateway) email address, provided by the client at registration time. Used for notification when the client (gateway) is approved.

The OAuth Client Approval Table contains the following rows:

- **oauth_consumer_key**: The OAuth client identifier, i.e., the primary key for the associated OAuth Client Table record.
- **approved**: A Boolean 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**: A timestamp entry for when the client was approved by TeraGrid staff.
- **approver**: The username of the TeraGrid staff person who approved the client (gateway).

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

The OAuth Transaction Table contains the following rows:

- **oauth_consumer_key**: The OAuth client identifier, i.e., the primary key for the associated OAuth Client Table record.
- **temp_token**: The OAuth temporary credentials identifier, generated by the OAuth server, which includes a timestamp for enforcing a limited lifetime on transactions. This is the oauth_token returned in the temporary credential request.
- **temp_token_valid**: A flag to indicate if the temp_token is valid (i.e., has not yet been used). Once the temp_token is used, the server marks it as invalid so it can not be used again.
- **oauth_callback**: An absolute HTTPS URL, provided by the client, back to which the server will redirect the resource owner (the user) when the resource owner authorization step is completed.
- **oauth_verifier**: The OAuth verification code generated by the server and sent from the server to the client via the user agent during resource owner authorization.
- **access_token**: The OAuth access token generated by the server for this transaction. This is the oauth_token returned by the server to the client in the OAuth token request.
- **access_token_valid**: A flag to indicate if the access_token is valid (i.e., has not yet been used). Once the access_token is used, the server marks it as invalid so it can not be used again.
- **certreq**: The certificate request submitted by the OAuth client (gateway) in the temporary credential request (see below).
- **certificate**: The certificate issued by MyProxy to be returned to the OAuth client (gateway) by the OAuth server when a valid access token is presented.

2.3 OAuth Client

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

The client API provides a very simple interface for science gateways. The gateway first calls a requestCertificate() function that initiates the transaction with the OAuth server and returns 1) a new, locally-generated private key associated with the certificate request, and 2) an HTTPS URL for redirecting the user's browser to the TeraGrid OAuth service. The gateway then redirects the user's browser to the provided URL, where the user authenticates at the OAuth service, and then the OAuth service redirects the user back to the gateway's configured HTTPS 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 URL as input to the function. The getCertificate() function completes the transaction with the OAuth server and returns the X.509 certificate for the user. As described above, the API handles local generation of private keys and certificate requests for the gateway.

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

- A science gateway operator enters the following information on the TGUP OAuth registration page:
  - **Science Gateway Name**: A friendly name for the science gateway for display to users on the OAuth Authorization page (see above).
  - **Science Gateway Home URL**: The home page URL for the science gateway for display to users on the 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 registration page then displays 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 TeraGrid OAuth server.
- The OAuth service notifies TeraGrid staff (i.e., the Science Gateways Area Director) who review the submission.
- If the submission is approved, a TeraGrid staff person runs a TGUP application to update the "approved" flag in the OAuth Client Table 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 TGUP OAuth registration page.

3. PROTOCOL

Figure 4 presents the protocol flow for the TeraGrid OAuth service. As the figure illustrates, the 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 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, obtaining a "temporary token" from the OAuth server, and then redirecting the user's browser to the OAuth server's authorization page with the "temporary token". The user then authenticates to the OAuth server using his or her TeraGrid (MyProxy) username and password and approves the gateway's request for credentials. The OAuth server submits the certificate request associated with the "temporary token", along with the user's username and password, in a GET request over TLS to the TeraGrid MyProxy server, which verifies the password and returns a short-lived signed certificate for the user if the password is valid. The OAuth server then stores the certificate and 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.

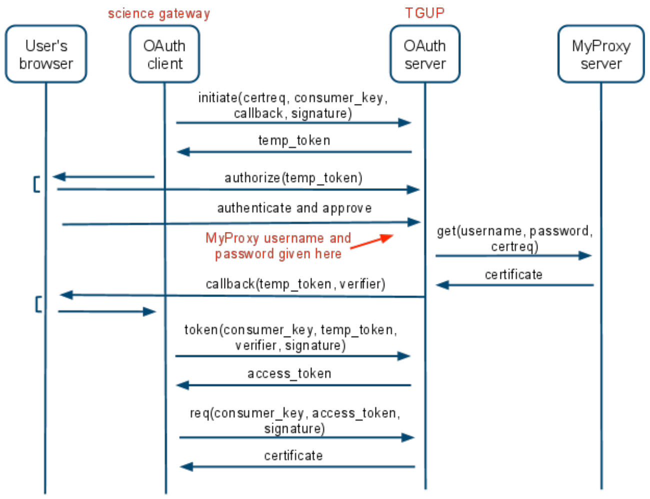

science gateway
TGUP
User's browser
OAuth client
OAuth server
MyProxy server
initiate(certreq, consumer_key, callback, signature)
temp_token
authorize(temp_token)
authenticate and approve
MyProxy username and password given here
get(username, password, certreq)
certificate
callback(temp_token, verifier)
token(consumer_key, temp_token, verifier, signature)
access_token
req(consumer_key, access_token, signature)
certificate

Figure 4. MyProxy OAuth Protocol Flow

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.

4. 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.
- **Focus on the web browser interface**: While OAuth may be used with other desktop applications besides web browsers, almost all TeraGrid gateways today provide a web browser interface. Non-browser applications are significantly more challenging from both a usability and security perspective. Therefore, we focus our initial work on the browser use case.

5. SECURITY CONSIDERATIONS

RFC 5849 identifies 15 security considerations for the OAuth protocol, many of which are addressed by our use of HTTPS. In this section, for sake of completeness, 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.

6. RELATED WORK

The OAuth protocol is widely adopted by web service providers today, including Twitter, Google, Yahoo!, Netflix, and MySpace. It is well suited to a variety of use cases for science gateways and grid computing. The Open Protein Simulator (OOPS) Science Gateway [15] uses OAuth with MyProxy for sharing certificates across gadgets that scientists assemble into a customized environment. The Open Life Science Gateway (OLSG) [14] uses OAuth to enable access to bioinformatics analysis gadgets via commercial OpenSocial sites such as iGoogle and has contributed the code to the Open Grid Computing Environments [7] platform for wider use. Other science gateways have identified OAuth integration as desired future work, including PolarGrid [16], QuakeSim [8], and the TeraGrid Visualization Gateway [6].

Grid certification authorities are also integrating OAuth into their services for issuing certificates to science gateways and other applications. Both the Confusa (confusa.org) and CILogon (cilogon.org) services provide OAuth interfaces similar to the TeraGrid OAuth service. The Confusa software enables the TERENA Certificate Service in Europe, while the CILogon service provides certificates to grid users in the United States. Both services support federated authentication for obtaining certificates, which is future work for the TeraGrid OAuth service.

7. FUTURE WORK

As discussed above, we have designed the initial version of the TeraGrid OAuth service with an emphasis on simplicity and ease of deployment, so we can provide this service quickly to TeraGrid gateways and learn more about how it is used and how it should be extended. In Section 4 we identify support for federated authentication and certificate renewal as two possible future enhancements, depending on interest and demand from TeraGrid gateways. We are also aware that the OAuth 2.0 protocol is currently under development in the IETF. When OAuth 2.0 is finalized and published as an RFC, we will consider updating the TeraGrid OAuth service to support it.

Adding support for federated authentication or other external authentication methods to the OAuth service would be relatively straightforward technically. TeraGrid already supports federated authentication using campus identity providers through the U.S. InCommon federation [4]. Federated authentication could be integrated with the "authenticate and approve" step in the MyProxy OAuth protocol flow (Figure 4), so that instead of providing a MyProxy password, the user would be prompted to authenticate with an external identity provider, and the OAuth server would verify the authentication assertion from that external provider. In this case the OAuth server would need to trust the external provider, and the MyProxy server would need to trust the OAuth server to properly authenticate the user, rather than providing a MyProxy password for authentication. Authentication with commercial identity providers, such as Google, could also technically be supported, if the TeraGrid project found those providers sufficiently trustworthy.

Migrating to OAuth 2.0 may facilitate support for certificate renewal. The current OAuth 2.0 draft specification includes refresh tokens, which allow an OAuth client to obtain a new access token from an OAuth authorization server when the current access token expires. This capability may map well to science gateway requirements for securely renewing credentials during long-running workflows, where the OAuth server would issue a refresh token along with the access token, allowing the gateway to obtain new certificates as needed, in a controlled and monitored fashion, using a succession of short-lived certificates and access tokens.

The OAuth capability we have developed for TeraGrid could have applicability in many other environments. For example, many other grid infrastructures around the world also use MyProxy to support web portal access, and TeraGrid science gateways often use other grid infrastructures in addition to TeraGrid. Building on our experiences with TeraGrid and CILogon, we plan to develop a general-purpose MyProxy OAuth package, without specific TeraGrid or CILogon dependencies, which can be deployed for other environments that use MyProxy. Additionally, we plan to explore other uses of OAuth for science gateways that do not specifically depend on MyProxy, such as authorization to REST services.

8. CONCLUSION

In conclusion, we have presented the newly developed TeraGrid OAuth service, integrated with the TeraGrid User Portal and TeraGrid MyProxy service, that provides certificates to science gateways. The OAuth service eliminates the need for TeraGrid users to disclose their TeraGrid passwords to science gateways when accessing their individual TeraGrid accounts via gateway interfaces. Instead, TeraGrid users authenticate at the TeraGrid User Portal to approve issuance of a certificate by MyProxy to the science gateway they are using. We have presented the design and implementation of the TeraGrid OAuth service, described the underlying protocol, and discussed the design decisions and security considerations we have made while developing the service in consultation with TeraGrid working groups and staff.

TeraGrid is now transitioning to “Phase III” under the NSF eXtreme Digital (XD) program. We expect the TeraGrid science gateways program to continue through this transition and to continue to grow into the future of the TeraGrid program, and therefore we expect the TeraGrid OAuth service to have long-term value.

Please visit http://security.ncsa.illinois.edu/teragrid-oauth for the latest information about the TeraGrid OAuth service, including protocol specifications and source code.

9. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under grant number 0932251.

We gratefully acknowledge input from the TeraGrid gateways and security working groups, as well as the TeraGrid user portal and Globus Online teams, during the development of this service. We also thank the anonymous reviewers for their recommended improvements to this paper.

10. REFERENCES

[1] E. Hammer-Lahav (ed.). The OAuth 1.0 Protocol. IETF RFC 5849 (Informational), April 2010. http://tools.ietf.org/html/rfc5849

[2] E. Rescorla (ed.). HTTP Over TLS. IETF RFC 2818 (Informational), May 2000. http://tools.ietf.org/html/rfc2818

[3] Jim Basney, Marty Humphrey, and Von Welch. The MyProxy Online Credential Repository. Software: Practice and Experience, Volume 35, Issue 9, July 2005, pages 801-816. http://dx.doi.org/10.1002/spe.688

[4] Jim Basney, Terry Fleury, and Von Welch. Federated Login to TeraGrid. 9th Symposium on Identity and Trust on the Internet (IDtrust), Gaithersburg, MD, April 2010. http://dx.doi.org/10.1145/1750389.1750391

[5] Jim Basney, Von Welch, and Nancy Wilkins-Diehr. TeraGrid Science Gateway AAAA Model: Implementation and Lessons Learned. TeraGrid Conference, August 2010. http://dx.doi.org/10.1145/1838574.1838576

[6] Joseph A. Insley, Ti Leggett, and Michael E. Papka. Using Dynamic Accounts to Enable Access to Advanced Resources through Science Gateways. Grid Computing Environments Workshop, 2009. http://dx.doi.org/10.1145/1658260.1658279

[7] Marlon Pierce, Suresh Marru, Wenjun Wu, Gopi Kandaswami, Gregor von Laszewski, Rion Dooley, Maytal Dahan, Nancy Wilkins-Diehr, and Mary Thomas. Open Grid Computing Environments. TeraGrid Conference, June 2009.

[8] Marlon Pierce, Xiaoming Gao, Sangmi Pallickara, Zhenhua Guo, Geoffrey Fox. The QuakeSim Portal and Services: New Approaches to Science Gateway Development Techniques. Concurrency and Computation: Practice and Experience, 22: 1732–1749. http://dx.doi.org/10.1002/cpe.1528

[9] Nancy Wilkins-Diehr, Dennis Gannon, Gerhard Klimeck, Scott Oster, and Sudhakar Pamidighantam. TeraGrid Science Gateways and Their Impact on Science. IEEE Computer 41(11): 32-41 (2008). http://dx.doi.org/10.1109/MC.2008.470

[10] Nancy Wilkins-Diehr (ed.). Science Gateways: Common Community Interfaces to Grid Resources. Concurrency and Computation: Practice and Experience, 19(6): 743-749 (2007). http://dx.doi.org/10.1002/cpe.1098

[11] Nancy Wilkins-Diehr and Thomas Soddemann. Science Gateway, Portal and Other Community Interfaces to High End Resources. ACM/IEEE Conference on Supercomputing (SC '06), 2006. http://dx.doi.org/10.1145/1188455.1188472

[12] T. Dierks and E. Rescorla (eds.). The Transport Layer Security (TLS) Protocol. IETF RFC 5246 (Standards Track), August 2008. http://tools.ietf.org/html/rfc5246

[13] Von Welch, Jim Barlow, James Basney, Doru Marcusiu and Nancy Wilkins-Diehr. A AAAA Model to Support Science Gateways with Community Accounts. Concurrency and Computation: Practice and Experience, 2006. http://dx.doi.org/10.1002/cpe.1081

[14] Wenjun Wu, M.E. Papka, R. Stevens. Toward an OpenSocial Life Science Gateway. Grid Computing Environments Workshop, November 2008. http://dx.doi.org/10.1109/GCE.2008.4738450

[15] Wenjun Wu, Thomas Uram, Michael Wilde, Mark Hereld, and Michael E. Papka. Accelerating Science Gateway Development with Web 2.0 and Swift. TeraGrid Conference, August 2010. http://dx.doi.org/10.1145/1838574.1838597

[16] Zhenhua Guo, Raminderjeet Singh, and Marlon Pierce. Building the PolarGrid Portal Using Web 2.0 and OpenSocial. Grid Computing Environments Workshop, 2009. http://dx.doi.org/10.1145/1658260.1658267