

# A VO-friendly, Community-based Authorization Framework

## Part 2: The Single Organization

*Ray Plante*

*NCSA*

*Version 0.1 (April 21, 2005)*

### Abstract

The era of massive surveys like LSST are driving the increasing necessity for astronomical research that is network-based and features remote data access and remote data analysis. With the development of the Virtual Observatory (VO) come the tools to make remote science easier. The VO community is large—thousands of potential users, and traditional authorization models based on individuals will just not scale. In traditional models, authorization policies are enforced solely using permissions associated with login accounts; thus, a user or group must exist for every set of authorizations. This three-part document proposes an application of the Globus Community Authorization Service (CAS) to centralize authorization policies that must be enforced by a number of resources, minimizing the authorization intelligence needed by the resources. To ease the burden on users, we introduce the concept of *weak certificates* that enable a sufficient level of access control common in many existing web based applications today but which is compatible with stricter grid security practices. In part 1, we describe three general use cases that we aim to address, list requirements, and summarize our approach using the Globus CAS. In part 2, we describe how the CAS-based model can be applied to a single organization that manages many distributed services and users within a single administration domain. In part 3, we extend the model for use in VO applications that span across administration domains; in this model, VO users can establish a single login that can be used with any compliant portal or service.

## 1 Introduction

In part 1, we introduced our vision of community-based authorization in terms of three broad use cases, a set of requirements, and an outline of the software components that make up the framework. In this part, we focus on the first two use cases, the PI-driven Observatory and the Survey-driven Observatory. In this part, we assume that we have a single organization that manages a collection of services distributed across different sites. The majority of users interact with the services through a portal; however, some or all might be accessible through custom clients (built either by the observatory or by someone else). These services are managed in a coordinated way, but since our model is based on certificates, we do not require or assume that there are common user accounts across all

systems. To help illustrate the context for this model, we will consider the specific case of the National Optical Astronomy Observatory (NOAO) in the examples. The application of this security model does not attempt to provide any interoperability with other portals; rather, it assumes its management of authentication and authorization is completely independent of any other portal. This application can be thought of as a transition toward the more general application that addresses interoperability across the VO as described in the part 3.

As outlined in section 3 of Part 1, our model is based on the use of X.509 certificates not only for enabling single sign-on authentication but also for transmitting authorization assertions. We expect that many of the services—particularly, the most application-specific ones—will be implemented as Web Services (using SOAP over HTTP) that support certificates via the WS-Security standard [ref]. However, a Web Service implementation is not a requirement. Other protocols, like GridFTP, perform better at transmitting large datasets; however, if they are used to access non-public data, they would need to support certificates. Other forms of authentication (e.g. ssh) could be used “internal” to the system; that is, for accessing a remote interface that is accessible to the user but rather which is accessed on behalf of a user with an “internal” account or identity.

Despite the use of certificates, we want the user experience to be similar to the many portals currently operating on the network that allow users to save personalized state and preferences. In such a portal, the user logs in once per session with a username and password. Perhaps only the expert users know that certificates are being used beyond the portal, but for the most part, the portal manages and uses the certificates transparently, without most users being aware. The key to this transparency is the storage of certificates on the server side and not on the user’s workstation.

Typically, when a user acquires a certificate from a commercial certificate authority (CA), the user downloads the certificate and the associated private key to her local workstation. She then might load it into her web browser. When a web site requires her certificate, the browser will typically engage in a dialogue with the user to authorize the use of the certificate. For example, with Firefox the following exchanges must occur (each generating its own pop-up window):

1. The user enters a password to access the internal database of certificates.
2. The browser asks which certificate the user would like to use
3. The user enters a passphrase that engages the use of the particular certificate.

Clearly, the overhead of first loading the certificate into the browser and then engaging its use is more complicated than a simple login page that most users are accustomed to. Most users (let alone astronomers) do not have experience using certificates within browsers; most, then, will not know how to access the interfaces for loading certificates. Of course, every browser will be slightly different and support will not be uniform. Recalling our first requirement (Pt. 1, 2.1-R1; see also comment regarding the patience of astronomers), this overhead will likely seem to users, at worst, unnecessary and, at best, not worth the trouble. Thus, in this model, we propose that the portal take responsibility

for storing certificates. The familiar user login process is used to “unlock” the certificate and engage its use.

## 1.1 The NOAO Context

NOAO represents something of an extreme example of a distributed environment. This observatory manages data from multiple telescopes in Arizona and Chili. Mirror sites for data are in the works at other sites such as NCSA. NOAO services could be deployed at any of these sites. As their facilities are PI-driven, the data are covered by a proprietary access period (typically 18 months), after which anyone may access it. NOAO is bringing on-line automated processing pipelines for a few of their current imagers, and it is expected that the generation of standard processed products will become commonplace in the future.

## 2 The Security Architecture Components

### 2.1 Authentication Management

The organization runs, through its portal, a set of four basic services to manage user logins and sessions:

1. *A registration service.* This is simply the form users fill out using a web browser in order to obtain a login to the system. This service collects the information about its users to establish their identities and their attributes which the organization needs to maintain the system. It also, of course, allows users to choose their login name and password (subject to the organization’s policies). The service will need to include mechanisms that assure, at some appropriate level, that the person who is filling out the form is indeed who they say they are (see section 3.1).

The portal will likely have to run a similar service that allows the user to update his registration information. In particular, it is common for astronomers to move to new jobs, so it will likely be important to portal administrators to keep institutional affiliations and email addresses up to date.

2. *A Certificate Authority (CA).* This service issues a certificate to users that successfully register—in particular, those that have provided sufficient evidence of their identity. The certificate is what declares the unambiguous identity of a user in the form of a *distinguished name* (DN), and as the signer of the certificate, the CA is root of trust in the assurance of that identity. All services accept require authentication will trust in the authenticity of the certificate a client presents because the services ultimately trust the CA that signed it. It’s important to note that this trust should not extend beyond the organization’s services; to do so requires more care in the authentication process (which is covered in Part 3).

The certificate issued by the CA will have a comparatively long lifetime—at least as long as the typical proprietary period (18 months), but perhaps longer.

3. *A Certificate Repository (MyProxy).* This stores the users’ certificates and associated private keys. An existing tool for doing just this that is widely used in

the grid community is called MyProxy [ref]. A username and password must be provided to in order to retrieve a certificate from the repository. The actual CA-signed certificate on its own is not actually returned, but rather a *proxy* certificate—one with a more limited lifetime (e.g., 1 day) and signed using the original certificate’s private key. Services trust this proxy because it can prove that the original private key was used to sign it, which links it to the original certificate signed by the CA.

4. *A login service.* This is another web page or portlet that prompts users for a name and password. Like any robust portal, this service will have to provide mechanisms for assisting users that have forgotten the password or login name, as it may be months or years between a user’s visits to the portal.

## 2.2 Authorization Management

### 2.2.1 The Community Authorization Service (CAS)

In our model, the centerpiece of the authorization management is the Community Authorization Service (CAS). In particular, we refer to the specific Globus (version 4) implementation as an exemplar of the functionality needed and of useful design choices. The CAS is, first, a centralized database of authorization policies. A web service interface allows detailed access to all objects that make up a policy; however, the most important interaction is one in which an individual’s identity (i.e. the user’s certificate’s distinguished name) is presented to CAS and a list of statements indicating what that user is allowed to do is returned. CAS encodes these statements using the Oasis standard Security Assertion Markup Language (SAML).

In the CAS data model, authorization policies—i.e. privileges to access restricted resources—are generally assigned to groups rather than users. This is useful when, as is typical with an observatories user community, users and their privileges will be highly dynamic: that is, new users will be continually added, individual users will be members of multiple projects, and new project collaborators will be added (and removed) all the time (see Pt. 1, section 2.2). In contrast to the users and their attributes, there will be fewer groups than users and their attributes will change less often; thus, it makes sense to assign policies to groups. Consequently, the CAS database also keeps track of defined groups and which users are assigned to them. Determining the privileges of a user, then, is a matter of assembling a union of the privileges assigned to each of the groups that the user is a member of. In detail, a CAS authorization policy has three components: an object or resource whose access to is being restricted, an action on the object, and a group that is allowed to carry out the action.

In the Globus Authorization Framework, there are two ways for a service to use CAS to enforce authorization policies. One is through a so-called “call-out” to the CAS server. Once the authentication process is complete, the service contacts the CAS to get the privileges, or *authorization assertions*, assigned to that user. In the other method, the client goes to the CAS server to retrieve the policies before visiting the service. Specifically, the client presents the user certificate to the CAS server and gets back (another) proxy certificate, called a *community credential*, which contains the privileges embedded inside. The user then presents the community credential to the service in lieu

of its user certificate. The server trusts the authorization assertions received in this way because the community credential is signed by the CAS server (which is linked via signatures back to the CA). This latter method is useful for portal-based access to the services because the portal can retrieve the assertions once and reuse them with each service it visits. This reduces the calls to the CAS service and eliminates a potential point of failure compared to the first interaction method. On the other hand, use of community credentials is less convenient when the service is being accessed from outside the portal, as from an external agent or client application since it places an extra hoop to jump through before the client can use the service. Thus, in general, both interaction methods will need to be supported.

## **2.2.2 CAS-enabled Services and Authorization Enforcement**

Of course, the other important component of the authorization management is the set of services that understand how to use CAS and community credentials to obtain authorization assertions. Once the assertions are acquired, the service enforces authorization by finding an assertion that applies to the client's request. If the appropriate assertion is not found, the request is denied. At this decision point, the service may fold in its own local policies into the decision, although for our single organization model, this would not likely happen.

How the service matches an assertion against a request depends on how the assertions are defined. At the extremes, they can either be defined to be highly service-specific or highly service-neutral. As an example of a service-specific policy, consider a GridFTP service that restricts access to specific collections and datasets. The object in a policy might be a directory or a file path and the action might be "read" or "lookup". This policy is very specific to this service because it specifically refers to real file paths. Consequently, it is very simple for the service to enforce this policy: it just does a string compare on the files or directories in the authorization assertions against the request. On the other hand, these specific assertions could not be used with another GridFTP server where the files might be stored with different names.

If the an authorization policy is defined in more general terms and not so connected with a particular service, then it is more easily applied to many services. The trade-off is that the service code will likely have to apply more intelligence to determine how the policy is applied to that particular service. As an extreme example, imagine that our policy object is "group t451 resources" and the allowed actions are "use" or "update". Now when a user requests a file from the GridFTP service, the service will have to lookup which group(s) the requested file is associated with. Perhaps a more "middle-ground" example would a policy object is a relative directory name and the actions are "read", etc. When a user requests a file, the service simply checks to see if the requested file is within a directory he has access to. In summary, how specific or general the policy is determines where the intelligence need to interpret the policy goes. Striking a good middle-ground will balance the number of policies that need to be defined against the ease of enforcing them.

As mentioned in the previous section, there are two ways to acquire the assertions, and in this model, a service would generally be prepared to support both. That is, the service would first look into the certificate to see if a SAML block exists inside it. If it is not

there, the service would extract out the user's distinguished name from the certificate and pass it to the CAS service itself to get back the assertions.

### **2.2.3 Certificate-enabled Client Applications**

There is at least one use case in which a portal user benefits from breaking out of the portal to download data (see scenario 3.X). This can still be accomplished without the user needing to explicitly deal with certificates through the aid of specialized client-side helper applications. This client application can launch itself as a traditional mime-type-triggered browser helper application or via Java WebStart. This application would receive a proxy certificate/private key pair from the portal which it can then use to access services independent of the portal. The proxy can include SAML assertions for handling authorization. Obviously, a certificate-enabled client application is not a critical component as most user sessions will not need this capability.

## **3 Scenarios**

In this section, using NOAO as our example organization, we illustrate how these components work together by describing specific scenarios that are critical to secure operation of restricted services.

For these scenarios, we assume a trust model that is very similar to what already exists at observatories like NOAO today. Observatory users provide contact information including affiliation in their proposals to use NOAO facilities. That contact information is then used to alert them when their proposals have been awarded time. That notification gives the user permission to use the facility. The act of preparing a successful proposal is considered sufficient evidence that the investigator is who she says she is.

### **3.1 How an Arbitrary User is Registered**

A user wishes to begin using NOAO services, either through the NOAO portal or through third-party client software. He may wish to access public data but make use of restricted services such as remote storage or customizable pipeline processing. Alternatively, he may expect to be added to an existing project group, thereby gaining access to that group's proprietary data.

1. The user accesses the portal registration page and fills out the form. The form includes the user's email address and institutional affiliation (if applicable) as well as the user's preferred login name and password.
2. When the form is submitted, the registration service validates and caches the registration information and then sends to the email address an automated message containing a URL that the user must visit to confirm and complete the registration. This URL includes (as an argument to the URL) a custom key string. By accessing this URL, the user demonstrates that he has put in her own email address.
3. The user visits the registration confirmation page. It may optionally present a visual test that proves that the page is being accessed by a real human and not a software agent.

4. Triggered either by the access to the confirmation page or the completion of the visual test, the registration service first stores the registration information into a user database. It next obtains a new certificate from the CA and deposits it (along with the associated private key) into the MyProxy certificate repository.
5. The registration service responds to the user that the registration has successfully been completed.

If the user plans to access restricted NOAO services via a third-party client, it will be necessary for her to download the certificate and private key and load it into the client. A user preferences page could be used to present an interface to do this. The PKCS12 format would be used for downloading the certificate.

We note that since this user has not gone through the proposal process, we don't have the same level of assurance his identity (see section 3); that is, he could have provided someone else's name and affiliation but his own email address. If this is considered a security concern, NOAO may elect not to afford as many privileges to this user until he appears as part of a proposal. The reduced status could be enforced by defining a special group (called, say, "investigators") that only includes users that have appeared on a proposal. Certain special policies for, say, running custom pipeline processing jobs, would be assigned to the "investigators" group. One might also claim that when a PI adds a "lesser" user to a proposal group, the PI is giving evidence that the person is who he says he is; thus, the user could be added to the "investigators" group at that time as well. These practices are a matter of policy and depend on how strong the trust model needs to be.

### **3.2 How a Project PI or CoI is Registered**

A new proposal has been awarded time on an NOAO telescope. When the data has been collected, the proposal's Principal Investigator (PI) and her Co-investigators (CoIs) will want to visit the NRAO archive and download the data.

1. When the PI creates the proposal, she provides her email address and affiliation to the "cover" form, along with those of her CoIs.
2. The proposal handling system extracts and stores the cover information into a proposal database.
3. When the proposal is awarded time, the proposal handling system notifies the registration service with a reference to the proposal information (including the PI and CoIs email addresses).
4. The registration service a new group is defined in the CAS database named after, say, the proposal's identifier.
5. The registration service searches the user database for each email address in the proposal. If the investigator's email address is found, indicating that she has already registered, then,
  - A.1. The registration service adds the investigator to the project group. If the investigator is the PI, then she is set as the group "superuser", allowing her to add other users who were not on the original proposal to the group.

A.2. The registration services sends an email to the investigator reminding her how she can access her data via the portal.

If the email address was not found,

B.1. The registration service sends an email to the investigator requesting that they create a login by accessing a special registration page. Included with the email is a custom key string that is randomly generated but unique to the investigator's email address.

B.2. The investigator accesses the page to find a form that requests the key string. By providing the string, the service is able to confirm that the user registering is the one awarded time on the telescope. The form also asks if the user has already created a login. If so, the investigator enters her login name and password, and the next step is skipped.

B.3. When the investigator submits the initial form, she is then sent to the normal registration page described in 3.1 (step 1), except this time many if not all the inputs have default values taken from her proposal cover form. She corrects the values and fills in missing information (like the desired login name and password) and then submits the form. The account set up proceeds just as in 3.1 (steps 2 through 5).

B.4. The registration service adds the user name to the new group associated with the proposal. If the user is the PI, then that user name is set as the superuser for the group.

The registration service needs to be prepared to handle a few human "failure" modes:

- The investigator may get several email requests to register, triggered by several successful proposals over several proposal cycles, before she gets around to registering. These requests can all include the same key. Once the login is established, the username is added to all the groups associated with the pending projects.
- The investigator may lose the email containing the key before she has a chance to register. For her to gain access to her proprietary data, she would need to contact the archive administrators who would manually trigger the resending of the registration request.
- Between the time the proposal was submitted and when the registration request goes out, the investigator has moved and changed institutions. If the email message is forwarded to the investigator's new institution, then the registration form can catch the change of email address. If the investigator does not receive the email request, establishing an account can be handled just as if the original request had been lost.

### **3.3 How a User Authenticates to Services via the Portal**

The user now has an account and is ready to use restricted NOAO services through the portal. The key feature of this scenario is the single sign-on per portal session.

1. The user accesses the login service through a web page. He types in his user name and password and then hits submit.
2. The login service passes the username and password onto the MyProxy certificate repository and gets back a proxy certificate (with an associated private key). The login service then sends this proxy to the CAS service and gets back a community credential containing the SAML authorization assertions. This certificate is cached with the portal for use through-out the session.
3. The user accesses a restricted service through a GUI interface—often referred to as a *portlet*—that is generated on the portal server rather than on the server that hosts the service. This allows the portal to use the community credential as the authenticating certificate on behalf of the user. Thus, the portlet connects to the service using the standard SSL protocol and the community credential. The portlet may harness a number of remote services, some requiring authentication and some not.

### **3.4 How a PI Adds/Removes Other Users to/from its Group**

The PI wants to allow her graduate student to access the proprietary data and related services associated with a particular project.

1. The PI instructs the student to create a login. The student follows the steps in Scenario 3.1.
2. Once the student has an account, the PI logs onto the portal (Scenario 3.3) and visits the portlet for managing groups and submits the request to attach the student's user name to the project's group.
3. The portlet uses the CAS Web Service interface and the PI's community credential (which authorizes her to add users to the group) to add the graduate student to the group.

A similar portlet would be used to remove the graduate student from the group if he flunks out of school.

### **3.5 How Authorization Policies are Defined and Used**

The PI's project gets scheduled on the telescope and begins generating data. The archive system will need to define the policies that protect it during the proprietary period.

1. When the PI's observing project is run on the telescope, the archive will assign the data generated to a collection specifically for that project, and that collection is assigned an identifier.
2. The archive system contacts the CAS service and defines a new policy object named after the collection identifier. This object is combined with the "read" action and assigned to the project's CAS group. This gives all members of the group read permission all data in the collection.
3. A Co-I accesses the portal to retrieve new data from the project. He authenticates according to Scenario 3.3. The community credential that that portal acquires from the CAS service includes an authorization assertion indicating that he has

- “read” permission on data from the collection having his project’s collection identifier.
4. The Co-I accesses a project browser portlet, chooses the datasets to retrieve, and starts the download.
  5. The portlet contacts the data retrieval service with a dataset request in which the dataset is referred to by its identifier. It passes with the request its community credentials.
  6. The data retrieval service first authenticates the user using the standard certificate-based mechanisms. It next extracts the authorization assertions and looks for policy objects that are collection IDs. For each such assertion found, the service tests to see if the file is part of the collection. If a match is found, the dataset is delivered back to the portal, which sends it back to the user.

A data retrieval service that delivers multiple datasets would benefit from parallel transfers. See next scenario for details on how this could be done securely.

### **3.6 How a Helper Application Securely Accesses Services directly**