CNL

Collaboratory.nl

Security in CNL

Version	: 1.4
Authors	: Aziz Ait Messoud & Redouan Lahit
Supervisors	: Cees de Laat, Yuri Demchenko
Institution	: UvA

Contents

1 Introduction

In the frame the Analytical Server Project at the Master education System- and Network Engineering of the University of Amsterdam. We have done a research about different authorization and authentication systems. And how they can be used in the CNL project

1.1 CNL

CNL (collaboratory.nl) is a project that is been launched by CNL consortium comprising of well known industrial company's. The project is aiming to make industrial collaboration possible. The goal of the project is open unique expensive laboratory equipment for external research and commercial use, and therefore provide secure collaborative environment for both customers and operators and experts. This means that different companies provide there resources to others, so they can use it for there own proposes.

CNL collaborative environment represent one of most complicated use cases for Authentication and Authorisation services. There is no available general purpose system with such range of functionality; however some components and solutions can be reused from existing solutions.

This document provides an overview of two open source software for Authentication and Authorisation that can be used in the CNL project.

1.2 The tasks in CNL

To understand how the CNL [1] works we analysed the CNL use cases to identify the major tasks and problems related to Authentication and Authorisation in this project, which we discus in this section.

1.2.1 Major scenario differences

To define the functionality that is needed in the CNL project there have been two scenarios used for this.

User scenario 1 is "Virtual Laboratory Unlimited" (VLU). This scenario deals with a situation when a customer gives an analysis assignment to a researcher. The researcher uses a remote instrument control and cooperates with other experts during the analysis. The results are remotely presented to the customer.

User scenario 2 is Watching Over The Shoulder" (WOTS). This scenario illustrates how customers and analysts can remotely interact during the sample analysis on the locally operated instrument.

The major differences between these two scenarios are:

- In scenario 2 the JobDescription is prepared at one site and experiment is run completely by resource/equipment owner/provider. In this scenario the local operator/analyst is in charge of operating the equipment.
- The operators are communicating with users via collaborative environment; customers are provided with intermediate or final results via Collaborative SW/environment.
- There may be need (for convenience) to have separate AAA service for internal purposes - for workers, and for external purposes - for customers; for some cases/orders there may not be need for customers to manage their own privileges/roles.

1.2.2 Scenario "Watching over shoulders" (WOTS).

Monday, analyst Peetra has a regular working day at DSM.
She receives a message from customer Bart of VitaLife: Dear Peetra we'd like to make use your expertise on topic xxx. We want to do some analysis concerning a product under development. Think we will need to use xyz-instrument for this. Could you please contact me and then we'll talk it over? Regards. Bart

Analyst Peetra phones customer Bart and does the intake of the assignment. She fills in the OrderWeb form to specify job characteristics end the sample that needs to be investigated. OrderWeb is the order tracking system of DSM. OrderWeb generates unique job number. Peetra provides a rough schedule of tasks to be performed to laboratory assistant Rob. Rob schedules the xyz-instrument in the DSM laboratory Ho checks with customer Bart whether the schedule is convenient for VitaLife since Bart and some colleagues will remotely attend parts of the research execution. I turns out that Wednesday 140 is convenient. The CustomerWeb session will start at 9:30 sharp.

CustomerWeb is collaborative environment that is linked to the laboratory instruments. Laboratory assistant Rob makes sure all necessary VitaLife people have an account. He sends them their account and login information. When the sample arrives, Rob registers it and prepares it for the research. He stores it in the sample cabinet near the laboratory.

On Thursday laboratory assistant Rob and analyst Peetra start early. At 8:00 AM they are present in the DSM laboratory. They load the sample. Zoom in and focus until they are satisfied with the picture they have. Peetra starts exploring the sample. At 9:15 Rob starts preparing the CustomerWeb session. VitaLife customers Bart and Greta login to the meeting at about 9:25. Audio and video connections are checked. At 9:30 the instrument is hooked up and analyst Peetra joins the meeting. Peetra replays her analysis. They discus the results and continue the analysis together. They ask Peetra to zoom in to other interesting areas. Every member has a personalized pointer that he she can use to point to areas of interests. At the end of the meeting VitaLife has an indication of what causes their problem. They decide that Peetra does some further analysis and that they will meet in 2 days. They want to invite manager Bert Douglas to join. Laboratory assistant Rob arranges a temporary account for Bert

The next day analyst Peetra sits in the lab and finishes the analysis. She prepares a presentation of the results and presents it in the Friday CustomerWeb session with VitaLife customers Bart. Bert and Greta. The VitaLife people discuss the results and the consequences for their new product. Peetra has an advisory role in the discussion. Every now and then Peetra is asked to do some additional analysis on certain aspects of the sample Al the end of the session the three customers thank Peetra for her advice and ask her to finish the report by next week.

2 Analysis of Authentication and Authorization requirements in CNL

2.1 CNL Authentication and Authorization requirements

2.1.1 Authentication

In the CNL system an authentication procedure to the users shall be enforced. Only registered users will/can use the CNL system. The users will be registered in a central registry. Therefore the users who want to register them self well be provided at least two options to authenticate themselves to the system using username and password or public key certificates obtained from trusted authority or confirming established (of-band) mutual trust relations.

In the CNL system it’s up to the instrument provider to accept or reject potential users according to a certain protocol. In cases when there is a need to preserve confidentiality of user/customer identity, authentication system should use pseudonyms or (trusted) authentication tokens.

2.1.2 Authorisation

Authorisation is essential for CNL as it is important to be able to restrict access to valuable resources and information within CNL. Therefore the CNL system shall grant or deny a certain action/privilege (such as access to a file, or joining a collaboration session) to authenticated users, based upon some policies and depending on the user attributes/roles linked/bound to user identity provided by the Authentication function.

To make management of access control easier, CNL incorporate Role-based Access Control approach using X.509 Attribute Certificates (AC) or XML-based SAML format.

2.2 Problem Description for Authentication and Authorization

In order to authorize access to a single Resource (e.g. information on a web server, e-learning application, library catalog etc.), the following generic interactions between a User and the Resource occur:

- A User (e.g. student, employee, library user, but also application) who wants to access a Resource has to register with the Resource Owner. The Resource Owner creates a virtual identity for this User, stores the necessary information about him/her and provides an identifier, like a login name, as well as credentials to the User. Later on, the User may use identifier and credentials to authenticate him-/herself to the Resource.
- The registered User wants to access a Resource and submits an access request to the Resource, claiming the virtual identity by identifying him-/herself with the identifier.
- The Resource asks the User to authenticate him-/herself (i.e. to provide the credentials belonging to that virtual identity).
- After checking the credentials, the Resource retrieves previously stored information about the User and,
- Based on this information, decides whether the access is authorized or not.

The disadvantage of this approach is that it does not scale if a User wants to have access to large numbers of different Resources:

- Registration: The User has to register with each Resource. This is the case even if many Resource Owners do not need to know the exact identity of a User (e.g., Resources which grant access to all students of a particular university only need to know if a User belongs to this university or not). The Resource Owner has to make sure that the information about its Users is always correct (e.g. to know if a person is still a student or not).

- Authentication: The Resources themselves may use different technologies to authenticate Users. Therefore, a User has to be able to handle different authentication technologies (e.g. password-based, certificate-based, smart-card-based, etc.) and for each technology at least one identifier, but often more than one.

To address this issue, larger organizations have recently started to implement local authentication and authorization infrastructures for their Users and Resources:

- Centralized User registration and storage of Authorization Attributes in a central User database
- Authentication and authorization interactions between User, Resource and central infrastructure of the organization

This approach solves the authentication and authorization issue for Resources belonging to the User’s Home Organization, but it is no solution for authentication and authorization across organizations.

- Users of one organization would like to be authorized to use Resources of other organizations (without having to register with each of these organizations).
- Organizations would like to open their Resources to (some) registered Users of other organizations in a controlled way, without having to register all these “foreign” Users by themselves.

2.3 User roles management for RBAC (role based access control)

In this section we analyse the VO concept for the purpose of user roles management, which is the requirement for RBAC

As we have seen Authorization plays a key role in the process of gaining access to resources in a computational grid. As for authentication, it is not feasible to administer authorization information on a local site basis, since users have normally direct administrative deals only with their own local site and with the collaborations they want to work in, but not, generally, with other entities.

It is convenient to introduce the following concepts:

- Virtual Organization (VO): abstract entity grouping Users, Institutions and Resources (if any) in a same administrative domain;
- Resource Provider (RP): facility offering resources (e.g. CPU, network, storage) to other parties (e.g. VO’s), according to specific “Memorandum of Understanding”.

From the authorization point of view, a grid is established by enforcing agreements between RP’s and VO’s, where, in general, resource access is controlled by both parties with different roles, and indeed the main difficulty is to clearly separate these two roles.

To solve this apparent dualism, we can classify the authorization information into the next categories:

- General information regarding the relationship of the user with his VO: groups he belongs to, roles he is allowed to cover and capabilities he should present to RP’s for special processing needs;
- Information regarding what the user is allowed to do at a RP, owing to his membership of a particular VO.
- Local policy of the RP which is applicable to the situation (site policy; optional).

The first kind of information should be contained in a server managed by the VO itself, while the second is probably best kept at the local sites, near the resources involved and controlled by some kind of (extended) Access Control Lists (ACL).

Users of a VO are organized in groups which in general form a hierarchical structure with the VO itself as the root; the management of a group can be delegated. Excluding the root, each group may have, in general, several ancestors. Users are normally contained in subgroups: for a user being member of a particular group G implies that he is also contained in all ancestor groups, even if it not explicitly stated, up to the root (i.e. in all groups contained in the paths from G to the root). Users are also characterized by roles they can cover in a group or at VO level (but in our model the VO is functionally equivalent to a group) and capabilities (properties to be interpreted by the local sites, e.g. ACL's). Roles are inherited by group members from ancestor groups (i.e. if a user as a role in a group and if he is member of one of its subgroups, he covers the same role in the subgroup), while the opposite is not generally true. The same inheritance rule applies for capabilities.

2.4 Basic AA/RBAC functionality required by CNL environment

In this section we describe the basic authentication/ authorization functionality that allows role-based access control (RBAC) based on specific attributes for an order or a job. Most of these functions can be related to AAA functions or AAA server.

- Initial input for this subsystem is provided from the JobDescription created from the OrderDescription, in particular, required information should include: job name/number and related approved and registered business agreement (BA); job category and related attributes, e.g. job credit; user list/identities and related attributes/roles/privileges

- Interaction with the user is provided via UserWebIF, interaction with the instrument – via ResourceIF

- UserWebIF and authentication services may include SSO (single-Sign-On) functionality to provide single user logon for a particular domain defined by BA or trust agreement (TA).

- Job/Order owner may possess RBAC admin functions that allow him/her to define and edit user accounts and roles/privileges

- Policy is defined by resource owner and has specific predefined profiles for different kind of jobs

2.5 Authorization Requirements

Authorization, as stated before, is based on policies written by VO's and their agreements with RP's that enforce local authorization. In general a user may be member of any number of VO's and his membership must be considered as“reserved” information.
On the other hand, a VO can have a complex structure with groups and subgroups in order to clearly divide its users according to their tasks. Moreover, a user can be a member of any number of these groups.
A user, both at VO and group level, may be characterized by any number of roles and capabilities; moreover roles and capabilities may be granted to the user indefinitely or on a scheduled time basis (e.g. a certain user of the CMS collaboration is granted administrative role only when he is “on shift”) or on a periodic basis (e.g. normal users have access to resources only during working hours).
The enforcement of these VO-managed policy attributes (group memberships, roles, capabilities) at local level descends from the agreements between the VO and the RP's, which, however, can always override the permissions granted by VO (e.g. to ban unwanted users). As a consequence, users must present their credential to RP's (and not just the authorization info).

3 VOMS

The requirements stated in the last section can be fulfilled by providing an entity (such as a user) with a certificate which both proves their identity, and proves the entity has membership and certain other attributes of a VO. Therefore DataGRID have decided to develop an EDG Virtual Organization Membership Service, or VOMS, to create this certificate for the user. In figure 1 we can see the steps how the certificate is created.

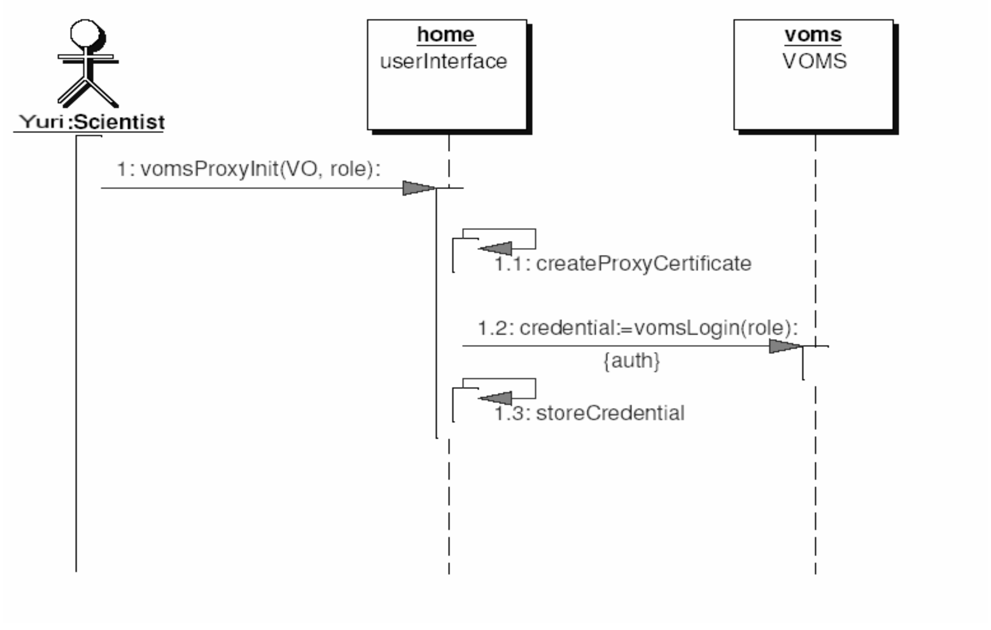
Yuri :Scientist
home
userInterface
voms
VOMS
1: vomsProxyInit(VO, role):
1.1: createProxyCertificate
1.2: credential:=vomsLogin(role):
{auth}
1.3: storeCredential

Figure 1: VOMS steps

The mechanism behind this kind of authorization is similar to the authentication system: a trusted service (CA in authentication; VOMS in authorization) signs a set of information (user's identity; user's groups) with its private key, thus its validity can be checked at the services using the public key of this trusted service. This new credential is included in the user's proxy certificate, which is signed by the user's normal certificate.

Since the VOMS certificate is a proxy the normal rules are applied: the lifetime of the authorization certificate can not be longer than the lifetime of the credential on which it is based on (typically 12-24 hours for a proxy certificate or Kerberos ticket).

3.1 The VOMS system

The VOMS system is composed by the following parts:

- Server: receives requests from a client and returns information about the user.
- User client: contacts the server presenting a user's certificate (or possibly proxy) and obtains a list of groups, roles and capabilities of the user.
- Administration Interface: used by the VO administrators (adding users, creating new groups, changing roles).

3.1.1 The server

The VOMS server is essentially a front-end to an RDBMS, where all the information about users is kept. The system is composed of the following parts (see figure 2):

- voms-proxy-init: the user client program, which contacts the server presenting a user's certificate and obtains a list of groups, roles and capabilities of the user.

- vomsd: the VOMS server, which receives requests from voms-proxy-init and returns information about the requester user.

- edg-voms-admin: command line interface for VO administrators (adding users, creating new groups, changing roles, etc...)

- admin GUI: graphical user interface for VO administrators

- web browser – web-admin servlet: web based administrative interface for VO administrators
- admin-server: the administrative server, which enforces the database consistency and the fine grained access control. This service is used by all of the administrative interfaces.

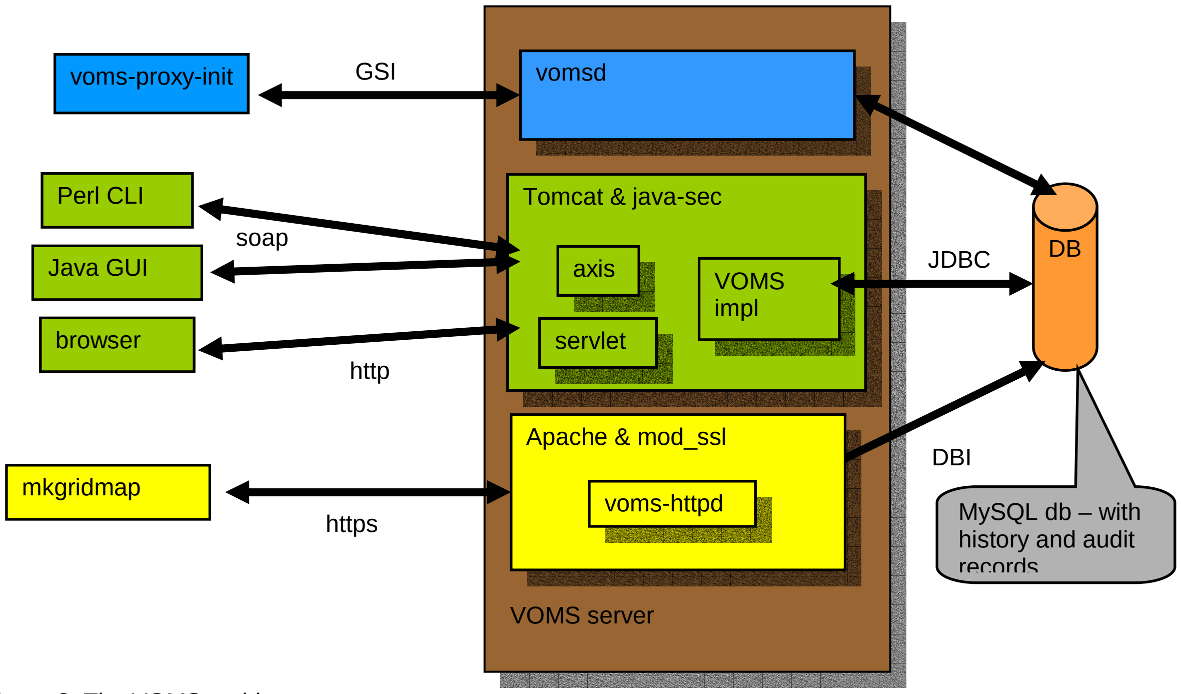

Figure 2: The VOMS architecture.

The VOMS architecture uses the authentication and delegation mechanisms provided by the Globus Toolkit Grid Security Infrastructure (GSI).

3.1.2 User

To make the creation of the user proxy certificate as quick and straightforward as possible for the user, the voms-proxy-init command has to be used in place of grid-proxy-init. This new command produces a user's proxy certificate – like grid-proxy-init – but with the difference that it contains the user information from the VOMS server(s).
This information is returned in a structure also containing the credentials both of the user and of the VOMS server together with validity. All these data are signed by the VOMS server itself. In the new release this well be an Attribute Certificate.

In order to get a certificate the next steps are performed:

1. Mutual authentication between client and server.
    a. Secure communication channel via Globus GSI.
2. The client sends request to server.
3. The server checks correctness of request.
4. The server sends back the required info in a "pseudo certificate" signed by itself.
5. The client checks the consistency and validity of the informations returned.
6. Steps 1-6 may be repeated for any number of servers.
7. The client creates a proxy certificate that includes the informations returned by the VOMS servers into a non critical extension.
8. Finally, the client may opt to include also additional information provided by the user.

The user may contact as many VOMSes as he needs.

In order to process this authorization information, the Gatekeeper, in addition to normal certificate checking, has to extract the additional information embedded in the proxy (the Pseudo-Certificate). This can be easily done with an appropriate LCAS plug-in [2]. However, as the VOMS info are included in a non critical extension of the certificate, this can be used even by "VOMS-unaware" Gatekeepers, thus maintaining compatibility with previous releases.

For the transition phase, DataGrid have also developed an enhanced version of mkgridmap (mkgridmap++) to allow RP's, with the old Gatekeeper installation, to query the VOMS server instead of the LDAP server. The access to VOMS is controlled by allowing only authenticated users via https.
The Java counterpart of LCAS/LCMAPS, the Authorization Manager is also capable of parsing and checking the VOMS Pseudo-Certificate and utilise its attributes in the authorization process. To ease the transition it is backward compatible and it is also able to use a grid-mapfile.

3.1.3 Service part – mutual authorization

To impose global decisions at the level of the virtual organization, the permissions have to be assigned to Grid services (e.g. resource broker, replica manager).
Since a service consists of many individual servers at the sites of real organizations, it will have many certificates proving the identity of the constituent servers. Managing all of these certificates in the accessed objects would be cumbersome, similar to managing large number of users. Therefore the services have to be assigned to the virtual organizations.

This means that a service must perform a similar "login" (voms-proxy-init) step to the user and thereby acquire a credential from the VO.

The benefit of this approach is that the client can authorize (accept) a service based on its attributes in a VO – mutual authorization beside the mutual authentication.

3.1.4 Administration

The administrative clients (web, CLI and GUI) share a common admin-server to modify the database. The server can be reached by the SOAP protocol so can easily be converted into an OGSA service.

The server consists of five sets of routines, grouped into services:

- Core, which provides the basic functionality for the clients;
- Admin, which provides the methods to administrate the VOMS database;
- History, which provides the logging and auditing functionality (the database scheme provides full audit records for every changes); all tables in the database have createdBy and createdSerial columns. The former contains the id of the administrator who created this record. The latter contains a database-wide unique, ordered serial number – incremented for each modification on the database – that identifies the operation (it is a transaction id); Request, which provides an integrated request handling mechanism for new users and for other changes; Compatibility, which provides a simple access to the user list for the mkgridmap utility.

Copies of deleted and modified rows are kept in a corresponding archive table. Archive tables have the same scheme as data tables, except for two additional columns: deletedBy, the administrator who deleted the record, and deletedSerial, the transaction number of the operation.

By keeping all expired data in the database, it’s possible to answer some questions in certain accuracy and efficiency. For example “Was user U in group G at time T?”.

3.1.4.1 Format of User Info Pseudo-Certificate

The signed information returned by the VOMS Server to the Client is composed by the following fields.

- Holder
  - Subject and Issuer of the certificate of the user requesting the info.
- Issuer
  - Subject and Issuer of the VOMS server certificate.
- Validity
  - notBeforeTime
  - notAfterTime

    For the moment it is an unique value for all the VOMS: if necessary it could become different according to the kind of info returned. Of course, the final decision is left to the local site.
- VO name
- Attributes
- Signature
  - VOMS Signature of the above data.

Future developments will include use of Authorization Certificates, replica mechanisms for the RDBMS, more sophisticated time validity for the VOMS certificates and subgroups.

For a complete overview about the attributes and the credentials that are needed we can refer to [4-5].

3.2 Summary

VOMS user or VO management interface can be used for user roles management in CNL.
One of the benefits of VOMS is how it deals with Attribute Certificates (AC). VOMS distributes these AC's to the users themselves, and this allows a much greater flexibility. For example, a user who is a member of several groups and holds several roles can actually choose how much information about himself he may want to present to a site. It is also possible to obtain and present ate the same time information on more VO's, a useful characteristic in case of collaboration between VO's.

Because VOMS does not focus on the policy issues (VOMS leave the interpretation of the AC's to other components i.e. LCAS) VOMS can be used in CNL for Authentication based on used belonging to some job oriented group or VO.

Because VOMS can record groups and roles, all the ultimate decision about what happens in a farm is kept at the Administrator farm. The farm administrator has total control about what happens. This makes the administration more efficient.

The VOMS server does not compromise the usual GSI security controls on the user's certificate before granting rights: it must be signed by a "trusted" CA, be valid and not revoked.

However even compromising the VOMS server itself would not be enough to grant illegal access to resources since the authorization data must be inserted in a user proxy certificate (i.e. countersigned by the user himself). Hence the only possible large scale vulnerabilities are denial of service attacks (e.g. to prevent VO users from getting their authorization credentials).

The main security issue about proxy certificates is the lack of a revocation mechanism; on the other hand these certificates have short lifetimes (typically 12 hours), which limits this vulnerability.

However, for the purpose of CNL it may be required to change the structure of user certificate, or use other formats for user attributes and roles like Attribute Certificates or XML based formats like SAML.

4 Shibboleth

4.1 Introduction to Shibboleth

Shibboleth is an initiative to develop an open, standards-based solution to the needs for organizations to exchange information about their users in a secure, and privacy-preserving manner. Shibboleth assumes that users make use of browsers and that the resources are accessible via standard browser technologies. Shibboleth is also a system that allows a user to choice in what information gets released about the user and to which site. Thus, the job of balancing access and privacy lies ultimately with the user, where it belongs.

The Focus of Shibboleth is on supporting inter-institutional authentication and authorization for access to web-based applications. The intent is to build upon existing heterogeneous security systems in use on campuses today, rather than mandating particular schemes like Kerberos or PKI based on X.509.

4.2 The Components

4.2.1 The Attribute Authority (AA)

Since Shibboleth is concerned with user privacy, an important element of the Shibboleth architecture is the component that releases information about users. This is the Attribute Authority (AA). Each User's Home Organization site has its own AA. The AA's job is to provide attributes about a user to a resource provider. But the AA also has the responsibility of providing a means for users to specify exactly which of their allowable attributes gets sent to each site they visit. Shibboleth architecture doesn't specify all of the implementation of the AA, it does specify a way for an Attribute Authority to structure "attribute release policies" (ARP) so that each user can choose what attributes get released and to where they get released. Usually AA's offer a web page that will allow a user to control attribute release policies for the sites they customarily visit, and to choose intelligent default policies for new sites. We'll discus Attribute Release Policies in Section 4.8.

4.2.2 The Share and it's interaction

In Shibboleth, when a user at a browser attempts to access a resource at a destination site (see step 1 in Figure 1), the "Shibbolized" web server will 'notice' that it doesn't have attributes about the user. The part of the web server that obtains and caches attributes is called the "Shibboleth Attribute Requester" (SHAR). The SHAR will then interact with the AA at the User's Home Organization site to get attributes about the user. The AA may store attributes directly, or more likely will obtain them from the institution's LDAP directory or other institutional database. Shibboleth doesn't specify how attribute release policies are stored and managed. That is up to each source site to decide on and implement. The AA must additionally have access to the user's "attribute release policy" for the destination site, in order to decide what attributes to send back to the SHAR.

The attribute request that the SHAR sends to the AA is called an AQM for "attribute query message". The response that the AA sends to the SHAR is an ARM for "attribute response message". The SHAR will send the attributes once it has these to the manager of the resource which the user is trying to access. The resource manager (RM) will then make an access control decision based on the user's attributes, and either grant or deny the user's request. If the user is simply trying to access a static web page or a typical application, this RM may be the web server itself.

In the case where the user is attempting a more complex action (say updating experimental results or transferring grant money), the RM may sit "behind" the web server on a separate machine.

4.2.2.1 The architecture in more detail

When a user contacts a destination, the SHAR would like to get attributes about the user. But on what basis can it ask? However Shibboleth allows for the case in which a user authenticates directly to a destination with a digital signature and client certificate (thus providing a name to the SHAR), its primary emphasis is on browser users without PKI credentials. Since Shibboleth wishes to obviate the need for a user to "log in" at each destination, and since Shibboleth is very concerned with privacy, this presents a dilemma: How can the SHAR ask for the user's attributes when it doesn't typically have the user's name?

Shibboleth solves this problem by associating a "handle" with the user. The handle lets a SHAR ask for attributes about the user, but the handle doesn't give away the user's name (or any other information that would identify the user to the destination). Shibboleth supports three different ways for a user without PKI credentials to become securely associated with a handled transmitted from the origin site to a destination.

- Direct Access to Destination Site
- Local Navigation or Resource Listing Sites
- Client Certificates

These three approaches are described below, each in their own sub-section.

4.2.3 Direct Access to Destination Site

A Shibboleth user will simply surf to a site they are interested in. In this case, the destination site doesn’t only not have attributes about the user; it doesn't have any identifying information. It doesn't even know what the user's origin site is. Before the SHAR component of the web server can ask for attributes about the browser user, the "handle" for getting information about this user needs to be obtained.

4.2.4 The SHIRE

The part of the "Shibbolized" web server that manages the process of acquiring a handle is called the SHIRE for “Shibboleth Indexical Reference Establisher” (See figure 1). "Indexical reference" refers to being able to "point at" a user without being able to otherwise name or describe the user. The web server has a "pointing" reference to that user. The SHIRE uses that connection with the browser to help initiate the process of securely getting a handle for the user. The SHIRE is co-located with the web server.

When a user first surfs to a Shibboleth-protected site, it is the SHIRE that takes over. The SHIRE's goal in this case is to determine the name and location of the user's Handle Service (HS), and ask it to send back a handle for the user. The Handle Service is responsible for making sure the user is authenticated locally at the User’s Home Organization site, and for creating a handle that can be used to retrieve attributes about the user.

4.2.5 Handle Extraction and SHIRE/SHAR Interaction

Shibboleth doesn't specify the interaction between the SHIRE and the SHAR components. In many, perhaps most, cases, the SHIRE and SHAR will be elements of a common implementation module within an HTTP server, allowing in-memory exchanges to take place in an obvious way.

What is specified, however, is the data to be communicated. The SHIRE must provide to the SHAR (or be certain the SHAR can obtain by other means) the following pieces of information for each request:

- the HTTP request URL, method, headers, etc. (i.e. the desired resource and the set of information which would naturally be expected by the endpoint of an HTTP request)
- an Attribute Query Handle (or an X.509 certificate serving as such)
- all AA binding information sent by the HS

- the domain name of the organization that issued the handle or certificate

A detailed description of Error Handling on the AQM/ARM and how the SHIRE uses the public key can be found in[3].

4.2.6 WAYS

The SHIRE uses an important helper that does most of the initial work. It is called the WAYS service ("Where Are You From?" service). The WAYF is a component that knows the name and location of the Handle Service for each client site that is participating in Shibboleth. Its primary job is to map an client site name (like UvA.nl) to the HS information for that site. The WAYF's other responsibility is to ask the user's HS to send a handle for the user to the SHIRE. Theoretically, the WAYF is a network based, possibly replicated service that lives "somewhere" in the Internet. However, in practice, WAYF information can be held locally by each SHIRE (for example, in a database, or even a file; the specifics are up to the destination site).

The SHIRE uses the WAYF by redirecting the user's browser to it. The WAYF will interact with the user, asking "Where are you from?" The user then enters the name of his or her institution. The WAYF provides the mapping between the name of the user's institution and the URL (and identity) of that site's HS. The WAYF then acts on behalf of the SHIRE and asks the HS to create a handle for the user and send it to the SHIRE. (This request is formally known as the "Attribute Query Handle Request" but we'll call it simply the "handle request"). The WAYF redirects the user's browser to the HS, appending the SHIRE's "handle acceptance" URL and the originally requested target URL as query string parameters.

4.3 Local Navigation or Resource Listing Sites (portal)

The previous section describes a user contacting a destination directly. An alternative form of contact is for the user to go through a "thin local portal" at the client site. That is, the user would surf to a page that contains a series of destination links. In reality, each destination link the user sees would be invisibly paired with the "handle acceptance URL" with which it is associated.

When the user clicks on a link, a handle request to the HS is made automatically via a redirection. This request has exactly the same format as a handle request coming from the WAYF: the handle acceptance URL and the user's desired destination URL are sent as query string parameters. The HS will authenticate the user if necessary, and then return a form containing the same sort of "handle package" that it would have, had the user accessed the destination first, as described earlier.

When the SHIRE receives the URL request with a handle package, it performs all of its impersonation checking and other validation work, and then passes on the handle and associated information to the SHAR. From the perspective of the SHIRE, there is nothing to distinguish an 'unsolicited' handle package from the result of a handle request it makes through the WAYF, because the SHIRE does not maintain state between handle request and response.

4.4 Client Certificates

A third scenario supported by the Shibboleth architecture is the use of a client certificate presented by the user to the destination site. This is distinct from the use of client certificates for local authentication; Shibboleth has nothing to say about this, since how a user locally authenticates to a HS is not specified by Shibboleth.

This scenario is quite a bit different from the previous two. In particular, since the user is directly authenticating to the destination, there is no need for the SHIRE to establish a handle-based context for the user (since this only applies to users who don't directly authenticate themselves to a destination). Also, because of the potential variation in certificate formats we don't yet have a standard mechanism by which the SHAR can obtain the necessary information from the certificate to build and send an AQM. At this point in time, origin sites that have certificate-bearing users will have to make agreements with destinations as to how information in the certificate should be used to form an AQM.

4.5 Review of the Shibboleth Flow

We now review the entire flow, from the user's initial contact of a destination through the SHAR receiving attributes about the user. The other use cases (portals, client certificates) are simpler subsets of this process. Figure 1 shows this flow. Here are the steps:

1. The user makes an initial request for a resource protected by a SHIRE.
2. The SHIRE obtains the URL of the user's HS, or redirects the user to a WAYF service for this purpose.
3. The SHIRE or WAYF asks the HS to create a handle for this user, redirecting the request through the user's browser.

4. The HS returns an opaque handle for the user that can be used by the SHAR to get attributes from the appropriate AA at the origin site. The arrow shows the SHIRE, after performing impersonation checks, passing on the handle (and AA information, and organization name) to the SHAR.
5. The SHAR asks the AA for attributes via an AQM message.
6. It receives attributes back from the AA via an ARM message.

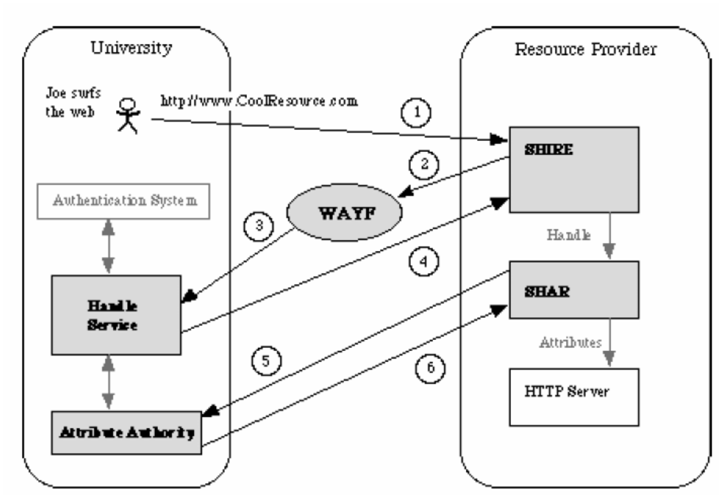

Figure 1: The Shibboleth Flow for "First Contact" case

4.6 HS and Multiple AA’s

Shibboleth doesn't specify how the HS creates a handle or how the HS knows the identity of the user. The HS creates the handle but only an AA uses it. Most HS implementations will first make sure that the user is logged into the client site's authentication system. The HS might also communicate the user's authenticated identity to the appropriate AA and even ask the AA to create the handle. In any case, both the HS and the AA must jointly understand that a given handle belongs to a particular user. No entity other than the AA (and the HS) can learn anything about the user from examining the handle alone. Thus, a username or the campus "id" of the user, while convenient, would not be a suitable handle.

Shibboleth separates the functions of handle creation and handle use. Shibboleth allows for the possibility of multiple AA’s at a given institution. Permitting multiple AA’s allows Shibboleth to be more easily adopted by institutions that have domains of separate administration. Each AA must be fully able to answer an attribute query about a user within its domain. There is no means within Shibboleth for a destination to be told to contact multiple AAs to retrieve attributes about a single user. The Handle Service at an institution with multiple AAs must be able to determine not only the user's identity, but which AA the user belongs to. How this is done is up to the specific implementation. See the Shibboleth Deployment Guide at[8] for additional guidance on handle creation and Multiple AA’s.

4.7 The user

The flows are relatively complex because Shibboleth tries to insure that the destination gets the right attributes about the particular browser user. This is much more difficult when there is no direct login by the user to the destination site. The important thing is that the user will be unaware of the complexity. In a fully-realized implementation, after the user speaks to the WAYF once (during a given browser session or for some defined period of time), the user shouldn't see any more Shibboleth-related pages. The flows between the destination, WAYF, and HS will still happen as the user surfs from destination site to destination site, but they will happen unnoticed by the user.

4.8 User attributes and policy

4.8.1 Attributes and Attribute Policies

Shibboleth's purpose is to securely transfer a user's attributes between the user's AA and the destinations the user wishes to interact with. But simply transferring attributes isn't enough: both User’s Home Organization and destination sites must agree on the attributes they will exchange and accept. Further, the User’s Home Organization site must allow users to choose which attributes get released and to whom, and the destination site must have a way of validating that the attributes it receives are legitimate.

4.8.2 Attributes

Generally, attributes in Shibboleth are name/value pairs, however richer structures are possible. The Shibboleth architecture doesn't specify a required set of attributes; however any practical use of Shibboleth will require a reasonably small set of “core” attributes be understood by all or most cooperating sites. We expect that in addition to a core set of attributes, pairs (or small groups) of sites will agree to additional attributes that have specific relevance. For example, two universities may decide that "Life Sciences Student" is an attribute that one university may send and that the other will receive and use in access control decisions.

4.8.3 Attribute Release Policies

Attribute release policies (ARP) are the rules that an AA follows when deciding whether or not to release an attribute to a requesting SHAR (see figure 1). A user may possess (and an organization may define) many different attributes; a SHAR asks generically for all attributes it is allowed to receive. An ARP defines the subset of attributes that the user and/or the organization wish to reveal to a particular SHAR for a particular resource.

An ARP at an AA consists of at least the following:

- A destination SHAR name
- Optionally a target URL tree
- A list of attributes that should be released to this SHAR and target URL pair

An ARP could also apply additional rules and guidelines based on just about any criteria an AA wishes to support, such as time of day, location, or many others. Each user may have as many attribute release policies as s/he needs. A fuller description of the different parts of the ARP follows:

4.8.4 A destination SHAR name

The destination SHAR name allows the AA to find the right ARP when a SHAR makes a request for attributes. The AA, in addition to having the name, also has (or can obtain) the public key certificate of the SHAR. Since attribute request messages are always authenticated, the AA can ensure that only legitimate requesters get attributes. (There is one exception to the "always authenticated" rule: the AA allows for the "anonymous" requester. An anonymous requester doesn't and can't authenticate.)

4.8.5 Optionally a target URL tree

The optional URL is for the case where a given SHAR may sit in front of very different subsites, or "application domains". An application domain is a set of resources that share the same resource manager. For example, recall that Professor Mary Smith was interested in both a multiinstitution research site and a multiple sclerosis site. And recall that she wished to release different attributes to each. The research site and the multiple sclerosis site constitute two different application domains. What if these application domains were both hosted at Ohio State? In that case, Mary's two ARPs would list "Ohio State" as the SHAR, but different URLs for the target.

4.8.6 A list of attributes

The list of attributes that should be released to the designated SHAR is maintained by the AA (perhaps in conjunction with the organization's directory). Note though that the user can only "release" attributes (and associated values) that s/he actually has. Thus a IT staff person, can release the attribute "staff", but cannot release "faculty" if he is not a faculty member.

Finally, note that from the user’s perspective, a SHAR is an alien concept, however, web browsers deal with hostnames and URLs, and most users are becoming at least somewhat comfortable with these notions. When a user asks that a given attribute be released (or not) to a particular target URL, the ARP that is being implicitly expressed is associated with a particular SHAR. The AA must be configured ahead of time to understand the association of URL targets and SHARs. A possible default association is for the SHAR name to be deemed to match the hostname in the URL.

4.8.7 Attribute Acceptance Policies

An attribute acceptance policy (AAP) is the flip side of the ARP; it protects the SHAR rather than the user or the AA. When a SHAR receives a set of attributes from an AA, it performs basic checks such as authenticating the sender (making sure it is the expected AA), expiration, signatures, etc. SHARs should also validate that each attribute is something that the AA can legitimately assert. Shibboleth, however, doesn't specify AAPs since what is acceptable depends on private agreements between institutions.

4.9 Cookies and it’s content

Shibboleth doesn't explicitly define a way of managing state or sessions for users, but the SHIRE MUST provide some means of subsequently identifying the browser user as "owning" the handle it acquired. Acquiring a new handle for each request (even if repeatedly getting the same handle back) is technically correct, but likely not acceptable because of performance. Session state will include the user's handle and all of the information that was associated with it i.e. home organization and AA location info. A typical session design would rely on locally scoped, nonpersistent HTTP "cookies", although other approaches are in use but not discussed in this report.

The handle itself is not secret, nor is it likely to be impractically large to store in a cookie, but there are various reasons why using the handle itself as a session identifier is a poor decision. Foremost among these is that the namespace of handles is not guaranteed to be unique, although it may be unique for a given HS. Additionally, since handles are not secret, it may be relatively easy to manufacture one that a SHIRE would accept as a valid session identifier, leaving the optional checking of IP address as the only protection against impersonation.

A much better solution would involve a more uniquely-generated token that relies on cryptography applied to the session data involved. For example, the handle, a timestamp, home organization and a random value maintained by the SHIRE could be used as input to a message digest function or encrypted with a secret key, producing a compact session identifier that is hard to forge but easy to verify. Such an approach would require the SHIRE to maintain an internal state table, rather than recovering all of the client’s contextual data from the cookie.

However once the token is associated with its state and issued to the client, the only practical protection possible remains IP address checking. The token can be made hard to forge, and the use of SSL may make it hard to steal, but the token may ultimately still be vulnerable at the User’s Home Organization, whose security is dependent on many software and human factors.

4.9.1 Pubcookie

Pubcookie [6] is an example of a "WebISO" package, a system designed to allow users, with standard web browsers, to authenticate to web based services across many web servers, using a standard, typically username/password central authentication service.
Pubcookie consists of a standalone login server and modules for common web server platforms like Apache and Microsoft IIS. Together, these components can turn existing authentication services into a solution for single sign-on authentication to websites throughout an institution. At[6] is information about the role Pubcookie can play in a Shibboleth system.

4.10 Summary

Shibboleth is a system for securely transferring user attributes from the User’s Home Organization (client side) to the site of the Resource Owner (server side), provided the Resource are accessible via standard web browsers. In addition Shibboleth enables the user to decide which information about them gets released to which site. The users therefore have to balance access and privacy. Shibboleth uses a federated administration, a Resource Owner leaves the administration of User identities and attributes to the User’s Home Organization, which is also responsible for providing attributes about a User that the Resource Owner can use in making an access control decision when the User attempts to use a Resource. Users are registered only at their Home Organizations, and not at each Resource.

Since Shibboleth is concerned with user privacy, an important element of the Shibboleth architecture is the component that releases information about users. This is the Attribute Authority (AA). Each client site has its own AA. The AA's job is to provide attributes about a user to a resource provider. But the AA also has the responsibility of providing a means for users to specify exactly which of their allowable attributes gets sent to each site they visit. Shibboleth architecture doesn't specify all of the implementation of the AA. it does specify a way for an AA to structure "attribute release policies" (ARP) so that each user can choose what attributes get released and to where they get released. Shibboleth doesn't specify how ARP’s are stored and managed. That is up to each source site to decide on and implement

In Shibboleth, when a user at a browser attempts to access a resource at a destination site (see step 1 in Figure 1), the "Shibbolized" web server will 'notice' that it doesn't have attributes about the user. The part of the web server that obtains and caches attributes is called the "Shibboleth Attribute Requester" (SHAR). The SHAR will then interact with the AA at the User’s Home Organization side to get attributes about the user. The attribute request that the SHAR sends to the AA is called an AQM for "attribute query message". The response that the AA sends to the SHAR is an ARM for "attribute response message". The SHAR will send the attributes once it has these to the manager of the resource which the user is trying to access. The resource manager (RM) will then make an access control decision based on the user's attributes, and either grant or deny the user's request.

The part of the "Shibbolized" web server that manages the process of acquiring a handle is called the SHIRE for “Shibboleth Indexical Reference Establisher”. The SHIRE is co-located with the web server. The SHIRE uses an important helper that does most of the initial work. It is called the WAYS service (“Where Are You From?" service). The WAYF is a component that knows the name and location of the Handle Service for each client site that is participating in Shibboleth. Its primary job is to map an client site name (like UvA.nl) to the HS information for that site.

Theoretically, the WAYF is a network based, possibly replicated service that lives "somewhere" in the Internet. However, in practice, WAYF information can be held locally by each SHIRE (for example, in a database, or even a file; the specifics are up to the destination site). Shibboleth architecture doesn't specify a required set of attributes, however any practical use of Shibboleth will require a reasonably small set of “core” attributes be understood by all or most cooperating sites. The JobDescription in CNL is an example of this set. The JobDescription contains all attributes necessary to launch and move the job through all required services to final completing. The choice of which attributes to use is left to implementers of CNL and other participants.

The important thing is that the user will be unaware of the complexity. In a fully-realized implementation, after the user speaks to the WAYF once (during a given browser session or for some defined period of time), the user shouldn't see any more Shibboleth-related pages. The flows between the destination, WAYF, and HS will still happen as the user surfs from destination site to destination site, but they will happen unnoticed by the user. Shibboleth doesn't explicitly define a way of managing state or sessions for users, but the SHIRE MUST provide some means of subsequently identifying the browser user as "owning" the handle it acquired. Shibboleth also does not implement Single Sing-On (SSO) as such, but may incorporate an existing SSO system. The architecture of Shibboleth is design for web-based Resources only this can be seen as an disadvantage.

Shibboleth uses Federative administration which is also an advantage. User’s Home Organization is responsible for the authentication, the User is responsible for selecting the data to be released, and the Resource owner is responsible for the authorization.

4.10.1 The pro’s and con’s of Shibboleth

One of the known pro’s of Shibboleth is the user privacy, User privacy was from the beginning part of the architectural design. The user controls which information gets released to which Resource. Shibboleth uses Federative administration which is also an advantage. User’s Home Organization is responsible for the authentication, the User is responsible for selecting the data to be released, and the Resource owner is responsible for the authorization. Further, Shibboleth is an open source software an has a good scalable architecture.

The architecture of Shibboleth is design for web-based Resources only this can be seen as an disadvantage.

4.10.2 AA Security Considerations

The attributes that an AA sends to a SHAR will be used to grant (or deny) access to resources, which might include research data, product information, grades, and other important information and services. Thus, these attributes are nearly as sensitive as user passwords. User’s Home Organization site administrators must ensure that administrative access to the AA and to the AA's source of attributes (be it a persistent store like a database or an LDAP directory, or something else entirely), is highly secured.

Poor security practices will increase the risk that the "wrong" attribute information could be associated with a user. Attributes that don't really belong to the user could be sent, thus empowering a user who shouldn't be empowered, or attributes that the user doesn't want sent (e.g. username) could be released, thus violating the user's privacy expectations.

This is not an complete list, but some "good practices" include the following:

- Auditing modifications to ARP’s (Attribute release policies)
- Auditing modifications to "meta policies" (e.g. policies about ARP precedence)
- Auditing modifications to the list of AA administrators and other roles
- Auditing modifications to attribute data, if appropriate and feasible

5 Conclusion and recommendations

5.1 Shibboleth and VOMS as architectural component in CNL

CNL Architecture is designed as open and configurable system that may use and integrate different pluggable functional subsystems. CNL user community is dynamic, defined by Job (or experiment) and may require interaction with user home organisation. The user Authentication system should be capable to use external authentication system from user home organisation or other Identity provider service.

CNL resources also can be distributed and integrated into “virtualised” collaborative environment. It is also important that component resources remains independent and continue to adhere to their local policies. Access control in CNL is based on roles assigned to user in the particular Job or experiment and should comply with the overall CNL policy and local policies at component resource sites.

CNL Authentication/Authorization should also provide administrative functions for user roles and attributes management defined as Role-based access control (RBAC) administration.

Based on CNL Security requirements and particularly Authentication and Authorization requirements, we can suggest that Shibboleth can provide Authorization functionality and user Authentication infrastructure; however Authentication service may be provided by external to Shibboleth system like WebISO/Pubcookie or VOMS. Required RBAC administration functions can be provided by VOMS.

As well VOMS and Shibboleth are able to handle a large and growing (scalability) number of users, resources and authentication/authorization services. The both systems are also very flexible; in they can be integrated with existing authentication schemes.

Shibboleth can use SAML for User attribute exchange. However, Shibboleth doesn’t handle policy issues such as automatic log-off. VOMS does not concern with this issue at all. The complexity of the Systems is manageable, the system components and interfaces are clearly defined.

In shibboleth we founded that the Vulnerability of the system is the WAYS. Because of its Server downtime and it is open to attack during redirects (SSL optional). Last Shibboleth release adds grouping of similar resources into “clubs” for easier handling.

The both systems are built on the same principals: Home Organisation handles authentication and the resource owner authorizes. Both in Shibboleth and VOMS, the User Home Organization manages User data. It stays the owner of the attributes. In Shibboleth is also possible for the user to decides which resources gets access to which attributes. User’s identity hidden behind a “handle”. Only user’s Home Organization can map a handle to a user.

The strength of both systems lays in there flexibility. This is because they are designed from the start to be used in a Grid environment.

Because Shibboleth can be used for as well Authentication as Authorization, we want to recommend it as a good solution for using in the CNL environment.
We only recommend the using of VOMS when it is used with other solutions. We think that VOMS can be used in a perfect combination with Permis [15]. VOMS as a AC issuer, and Permis as an policy engine.

5.2 Shibboleth & VOMS Requirements

The implementation requirements of Shibboleth are:

- Apache v1.3 and on the Home Organization side Tomcat.
- LDAP server for storing user information and attributes
- Clock synchronization required (NTP on all servers)
- Client browser must support redirects and SSL, JavaScript is preferably

The implementation requirements of VOMS are:

- GSI version 2.0 or higher.
- Database used: MySQL >= 4.0.13
- Java 1.4.x(soap interface)
- Tomcat 4
- edg-java-security
- Apache (or any other http server)

- Perl 5. (The same as plain mkgridmap plus a few more modules)

5.2.1 Which attributes required in CNL?

To let the both systems work properly, registered business agreement have to be made between participated organisations about how and what (which attributes) to use for communication. At all times it must be clear for both sides which attributes may not be sent, and which are been accepted.
As we discussed earlier the both architectures doesn't specify a required set of attributes, however any practical use of Shibboleth will require a reasonably small set of “core” attributes be understood by all or most cooperating sites.

The JobDescription in CNL that defines both Job related attributes and user roles are an example of this set. The JobDescription contains all attributes necessary to launch and move the job through all required services to final completing. The choice of which attributes to use is left to implementers of CNL and other participants, however CNL will provide administrative interface for the Job user attribute management. For example, two organisations may decide that "New Design" is an attribute that one organisation may send and that the other will receive and use in access control decisions.

5.2.2 Single Sign-On (SSO) in CNL

In the current CNL architecture we notice that UserWebIF and authentication services may include SSO functionality to provide single user logon for a particular domain defined by registered business agreement (a Job description attribute), in fact each interaction with the user is provided via UserWebIF.

Authentication method in Shibboleth is left to the User’s Home Organization. It does not implement SSO as such, but CNL wil have few options of using native Shibboleth WebISO/Pubcookie system or VOMS, or additionally use user local Authentication system. In the Shibboleth, the HS receives the handle request as a set of parameters from the user's browser; it must at this point determine who the user is. Shibboleth doesn't specify how this is accomplished; the choice of authentication method is left to the User’s Home Organization site. Logically, the HS can be thought of as a local application that requires authentication for it to operate properly. This can be accomplished by whatever means are locally acceptable, be they name and password, client certificate, a customized single sign-on interaction with cookies, or something else.

The HS may or may not be a component of a User’s Home Organization site's implementation of single sign-on across servers and applications. It may perform authentication itself or it may delegate this function and perform its task following that authentication. It may remember the user's identity in order to implement single sign-on across Shibboleth destination sites, or it may choose to require the user to authenticate each time a new destination requests a handle. These decisions are left to implementers.

6 References

[1] = http://www.telin.nl/Middleware/Collaboratory/
[2] = Architectural design and evaluation criteria: WP4 Fabric Management, DataGrid-04-D4.2-0119-2-1 (2001)
[3] = http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v04.pdf
[4] = http://edg-wp2.web.cern.ch/edg-wp2/security/voms/edg-voms-credential.pdf
[5] = European DataGrid, Security Coordination Group: Security Design, DataGrid-07-D7.6-0112(2003)
[6] = http://www.pubcookie.org/
[7] = http://www.contrib.andrew.cmu.edu/~leg/work/library-kiosk.html
[8] = http://shibboleth.internet2.edu
[10] = http://shibboleth.internet2.edu/release/shib-v1.1.html
[11] = http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v04.pdf
[12] = GGFSchool-new.ppt
[13] = VOMS-v1_1.pdf
[14] = XML Encryption Syntax and Processing W3C Recommendation 10 December 2002 - http://www.w3.org/TR/xmlenc-core/
[15] = Privilege and Role Management Infrastructure Validation: http://www.permis.org