GAMA: Grid Account Management Architecture

Karan Bhatia, Sandeep Chandra, Kurt Mueller
San Diego Supercomputer Center
{karan,chandras,kurt}@sdsc.edu

Abstract

Security is a critical component of grid systems and while there are numerous software components and tools that provide some capabilities relating to security, there are few complete end-to-end security systems that work for emerging grid infrastructures "out-of-the-box". Hence significant time and effort are needed for software evaluation, testing and integration of different software components despite the fact that most grid infrastructures have very similar requirements and needs. In this paper, we present GAMA, a GSI-based security infrastructure that is easy to deploy and supports multiple applications. GAMA consists of two component groups: a backend server component that creates and manages X.509 user credentials on behalf of the user, and a set of portal components that provide interfaces for both users and administrators. The GAMA server provides synchronization capability between portals and clusters, supports any SOAP-based rich clients, and supports legacy applications that rely on MyProxy.

1. Introduction

Security is a critical component of "grid" systems [1], which are distributed systems that span multiple organizational and administrative domains. The standard technology for security in government and academic grids is the Globus Security Infrastructure (GSI) [2]. GSI is a public-key-based, X.509 [3] conforming system that relies on trusted third parties for signing user and host certificates. Typical usage models require that each user is assigned a user credential consisting of a public and private key. Users generate "delegated proxy" certificates with short life spans that get passed from one component to another and form the basis of authentication, access control and logging.

Despite the general acceptance of GSI and its use over the course of many years, GSI-based security systems are well-known to be difficult for administrators to deploy and for users to use. Various grid systems have approached this problem differently. For instance, many projects build certificate management capabilities directly into a web portal [4-6]. While this does provide a simple interface for end-users, it limits the types of applications with which the user can interact to those accessible from that specific web portal. Other projects provide tools for the end-user to manage their own credentials, but in many cases, these tools are different for resources in different administrative domains and require a steep learning curve for the end-user. From the grid systems architect's point of view, while there are a number of tools that support building a GSI-based security infrastructure, these tools provide different and overlapping APIs and are not designed for interoperability. For example: there are a number of different software packages for building "certificate authorities," the trusted third parties that sign and manage user certificates, including CACL [7] and simpleCA [8]; storing certificates is accomplished by another software component called MyProxy [9, 10]; certificate management is sometimes left to users and other times automated by a web-portal; authorization can also be performed in a variety of ways, from the traditional GSI gridmap files that map user certificates to local system accounts, to the use of Akenti [11] and Security Assertion Markup Language (SAML)[12, 13]; role-based access control may be provided by the Community Authorization Service (CAS) [14, 15] or Virtual Organization Management (VOM) [16].

We believe a flexible approach that leverages a services-oriented architecture and aggregates existing trusted tools can address many aspects of the security infrastructure problem. In this paper we present GAMA, a Grid Account Management Architecture developed jointly by the GEON [17] and Telescience [18] projects. While the goals of the two projects are significantly different (the GEON project provides distributed data querying capabilities across datasets hosted by fifteen partner sites while the Telescience project integrates access to rare and globally distributed bio-imaging instruments and to computational and data storage resources), both projects have remarkably similar security requirements. We summarize the requirements here:

1. It is unreasonable for most users to install middleware components on their desktop or laptop client systems. The typical user in both projects wants to access the grid systems through a web or grid portal with no grid software installed and running on their machine. Users want a familiar username/password model for logging into portals. They don't know or care to configure various middleware (MyProxy, CAS, etc). Users also don't want to manage certificates or create proxy certificates themselves. Only in certain unusual cases would a user require access to their actual certificate or proxy.
2. There are multiple types of resources in each project, and for each type of resource, multiple instances spread geographically in different administrative domains. Resources include web portals, clusters, databases and domain-specific applications. For example, the GEON project specifies a "distributed portal architecture" that allows each partner site to build a local "satellite" portal that is customized for the users at that site while still providing access to global GEON resources. Access to all project resources by a user should be enabled with a single username/password pair identifying that user.
3. Role-based authorization mechanisms provide authorization based on the identity of a user and the role to which the user is assigned. Ideally, no gridmap files or local accounts should require updating as users are added or deleted from the project. While authorization for resource access is done through a user's role, detailed logging/accounting is based on the user identity.
4. The grid security infrastructure of the projects must support a wide variety of applications, including: web-based applications such as web portals, Java Web Start, and Flash; rich or thick clients, such as stand-alone applications; and traditional command-line tools such as ssh.

GAMA provides an "out-of-the-box", complete, end-to-end, GSI-based security infrastructure that supports multiple web portals, clusters and a wide variety of end-user applications. It is currently being deployed for production use in the GEON and Telescience projects and is being investigated for use in a variety of other projects. Version 1.0 has been released and is available for download[1]. GAMA consists of two components: a back-end GAMA server that manages users' identities and credentials, and a set of web portlets that provide the main interface to users and administrators for credential management. The back-end services use CACL for creating credentials, MyProxy for storing credentials and CAS for defining and using roles. These services are wrapped with standard web services mechanisms and are accessible through WSDL interfaces by any programs or environments that support such mechanisms. In Section 2 we discuss the interfaces on the GAMA Server back-end. In Section 3 we discuss the capabilities of the web portlets which provide a number of configurable interfaces for end-users to request and use accounts, and for administrators to define policies and manage users. In Section 4 we discuss other clients, including a command-line client for providing cluster management and rich clients for various other projects. In Section 5 we provide a brief evaluation, and we conclude with a comparison to similar systems in Section 6.

2. GAMA server

The GAMA Server consists of a set of services that run on an isolated, locked down machine with no end-user access. The GAMA back-end services run on this machine and are accessed solely through web-services calls. These services provide a consistent interface to the various portal clients, providing basic account management and login/logout capability. Behind the scenes, the services manage the user's certificate and private keys and interface with CACL, MyProxy and CAS, as needed, depending on the configuration and the type of account management system an organization decides to use. CACL provides an implementation of a "certificate authority" that generates user and server certificates; MyProxy provides a centralized certificate repository with advanced features such as certificate renewal; and CAS defines a set of roles and a set of access rights for each role, and maps users to roles within the GSI framework.

GAMA defines the following services running in two different web services containers, one that requires mutual authentication between the GAMA Server and GAMA clients, and one that does not. The SecurityCACLService, SecurityMyproxyService, and SecurityCASService run in the more secure web service container and provide functionality to create and delete users. The SecurityMyproxyloginService, SecurityCASloginService, and SecurityUserImportService run in the less secure container (but still run over a secure communication channel) and provide basic login and user query capability.

[1] See http://grid-devel.sdsc.edu/gama for download information.

The SecurityCACLService provides access to the CACL CA functions. To create a new user, `createUserCACL()` method is invoked on the authentication service with the email address of the user, the full name associated with the email address and the password with which to encrypt the user's newly-created private key. In order to delete users and their associated certificates, the `deleteUserCACL()` operation is invoked with the email address identifying the user. The service will invalidate and delete the associated user certificate using the appropriate CACL scripts.

The SecurityMyproxyService defines methods that manage user credentials in the MyProxy repository. To create a new user `createUserMyProxy()` is invoked which calls a MyProxy wrapper class that loads the credentials of the user identified by email to the MyProxy repository location from the CACL certificate repository. In order to delete users and their associated credentials from the MyProxy repository, the `deleteUserMyproxy()` operation is invoked with the email address identifying the user. The operation invokes a MyProxy wrapper class that removes the credentials from the MyProxy repository.

The SecurityMyproxyloginService is invoked when the user logs into a portal or GAMA-enabled application. This method uses the MyProxy server running on the GAMA Server to generate a user proxy certificate, which is returned as a String.

A fourth interface, SecurityCASService, provides operations that implement CAS functions. The `createUserCAS()` operation enrolls the user into the default role in the CAS database. Similarly `deleteUserCAS()` deletes the user information from the CAS database.

The interface SecurityCASloginService is invoked when the user logs into a portal or GAMA-enabled application. The login operation returns a String representing the user proxy certificate. This method first invokes a wrapper class around the Java COG toolkit to run grid-proxy-init for the user trying to login. It then invokes the CAS service to run cas-proxy-init on the user proxy to generate a CAS proxy. This CAS proxy adds assertions to the original user proxy. Multiple logins will return the same proxy certificate so long as the certificate is valid. The `logoutUserCAS()` method is invoked when a user logs out of the portal. This method simply calls the grid-proxy-destroy command. This is optional, as the user may never log out. The proxy certificate issued when a user logs in is valid only for a short time period, and if the user never explicitly logs out, the certificate will expire. Nevertheless, on logout, the proxy certificate is destroyed.

The final interface is the SecurityUserImportService. This interface provides functionality to support the distributed portal architecture model within GEON. This operation returns a list of valid users on the back-end server. An admin at a satellite portal can invoke this method to add users, whose credentials exist on the back-end server, to their local portal database. This feature eliminates the need to locally add users on each of these satellite portals if their credentials already exist on the back-end GAMA server.

Security between the back-end services and the portal is achieved using secure http (https) and mutual authentication. On the GAMA back-end server some of the interfaces may only be invoked by trusted parties (such as portals) and therefore the GAMA backend server and portal server need to mutually authenticate each other. In particular, interfaces involving account creation or deletion like SecurityCACLService, SecurityMyproxyService and SecurityCASService should only be invoked from trusted portals and therefore have to be mutually authenticated. Interfaces that implement user login and import need not require mutual authentication as they could be invoked by clients such as Java Web Start and Flash applications.

We use the CA system to generate appropriate host certificates, a dedicated Apache Tomcat instance to serve secure services, and the standard Java keystore mechanism for certificate storage. We achieve mutual authentication in the system by adding the host certificate of the portal server as a trusted certificate on the GAMA back-end server's keystore and adding the host certificate of the GAMA back-end server as a trusted certificate on the portal server's keystore. Services that do not require mutual authentication are served by a separate Tomcat instance running on a different port.

The backend GAMA server requires services and software packages, including CACL, Globus, MyProxy, CAS, web service libraries (Axis) and services, Apache Tomcat servlet containers and other dependencies. These software packages and services can be difficult to install and some require a substantial amount of configuration after they are installed on the server, which is cumbersome for a system administrator. To make this process easier we have created a GAMA server roll [19], which is based on the Rocks [20] clusters management software model. The GAMA roll modularly plugs into the Rocks framework and is used to install all software packages used by the GAMA back-end server and perform post install configuration. When installing a GAMA backend server the admin only has to provide information to configure the CA system, such as organization name and organization unit. The Rocks software then installs the basic operating system and all the components of the GAMA roll. During the first boot of the system it performs the post install operations that include, among many operations, configuring the CA system, configuring Globus, MyProxy and CAS servers, deploying the web services to the appropriates servlet containers and starting all the necessary services using startup scripts. This way we provide a fully functional GAMA back-end server running out of the box with minimal effort on part of the system administrator.

3. GAMA portlets

The second component of the GAMA system consists of a set of portlets that provide an interface for users to request an account, login and access web-based applications, and provide administrators the ability to easily define policies and perform user management tasks. The GAMA portlets provide the following features:

- end-users can request grid-enabled accounts and provide the necessary information for the administrator to determine whether the user should be accepted as a grid user,
- once accepted, users can log in as with any other commercial web site using their email address and password that they supplied,
- the system provides email verification before accounts are accepted,
- administrators can define a set of rules for automatically accepting or rejecting user account requests based on information provided,
- administrators can configure the system to provide temporary "local" accounts to the user while their request is pending,
- the system provides a mechanism for "satellite" portals to import accounts previously created.

The GAMA functionality is available as a set of JSR168-compatible [21] portlets that run in the GridSphere [22] portal container.

3.1 User account request process

The user account request process is modeled after common practice in commercial web sites: users go to a web site and click a link to request an account. On the account request page, the user fills out a form and provides name, contact information, and institution name, as in Figure 1. The specific set of information collected per user is easily configurable.

Once the form is submitted, the information is processed by the system and, if appropriate account acceptance rules are defined, may be automatically accepted or rejected. If no rules are defined, the information is stored for the administrator to manually approve or reject based on the information the user provided.

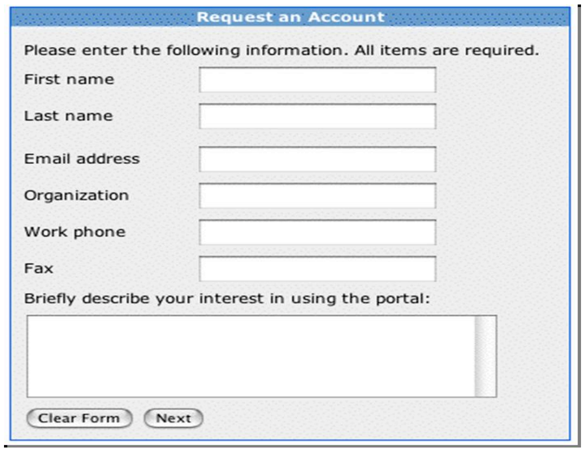

Request an Account
Please enter the following information. All items are required.
First name
Last name
Email address
Organization
Work phone
Fax
Briefly describe your interest in using the portal:
Clear Form
Next

Figure 1: Account request form.

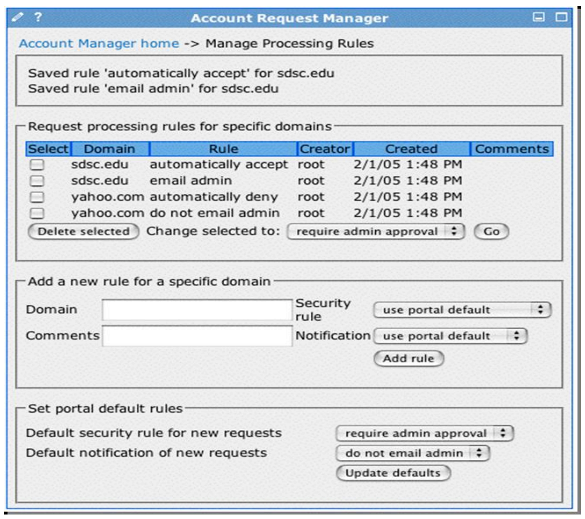
If the account is approved, the system sends an email to the user's email address with an embedded URL activation link in order to ensure that the email address is valid. The activation link contains a random element to ensure that it can not be easily guessed. The user activates the account by visiting the URL activation link. At this point, the system asks the user for a password that it will use for encrypting the user's private key. Once provided, the portal contacts the GAMA server that creates the user account. Note that the password is never stored by the portal or the GAMA server. If at any point in the process an error occurs, debug information is stored for the administrator to review and all changes to the server are rolled back.

3.2 User login process

After the account request is approved and the account is created, the user can login to the portal using the email address and password that she provided earlier. During login the portal passes the username and password of the user to the GAMA Server through the login service, which tries to retrieve a limited-lifetime proxy from the MyProxy server running on the GAMA Server with the supplied username and password. If MyProxy retrieval is successful, the login service returns the proxy for the user back to the portal. The user is then logged into the portal and their proxy is activated in the GridSphere portal environment for use by grid applications running there.

3.3 Managing account requests

The GAMA portlets allow the administrator to define rules about who is approved for accounts based on the domain of the email address of the user. Figure 2 shows an example configuration of the acceptance rules in which the administrator has configured the portal to automatically approve account requests from users within the sdsc.edu domain and send the administrator an email message when such users apply. Users with yahoo.com email addresses are automatically denied. By default (if no rule matches), the portal requires the administrator to approve the account request by hand though this default behavior can also be changed.


Account Request Manager

Account Manager home -> Manage Processing Rules

Saved rule 'automatically accept' for sdsc.edu
Saved rule 'email admin' for sdsc.edu

Request processing rules for specific domains

Select	Domain	Rule	Creator	Created	Comments
☐	sdsc.edu	automatically accept	root	2/1/05 1:48 PM	
☐	sdsc.edu	email admin	root	2/1/05 1:48 PM	
☐	yahoo.com	automatically deny	root	2/1/05 1:48 PM	
☐	yahoo.com	do not email admin	root	2/1/05 1:48 PM	

Delete selected Change selected to: require admin approval Go

Add a new rule for a specific domain

Domain
Comments
Security rule: use portal default
Notification: use portal default
Add rule

Set portal default rules

Default security rule for new requests: require admin approval
Default notification of new requests: do not email admin
Update defaults

Figure 2: Administrator interface to define account acceptance policies.

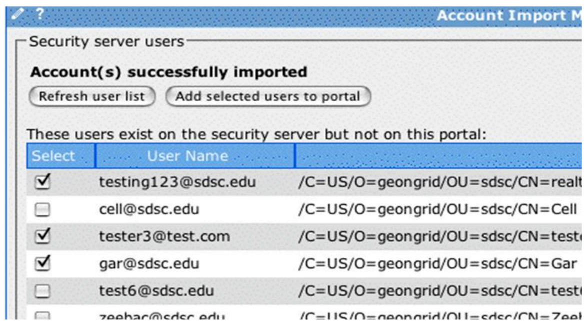
In addition, the administrator has the capability to allow "local user" access to the portal while the account request is being processed. Such local users do not have grid credentials and do not have any information stored on the GAMA server and can not be imported to other resources (such as portals, clusters, etc). The purpose of the local user feature is to enable new users to get started using the portal immediately, and portal applications running within a portal can determine that a user accessing them is a "local" user rather than a grid user and provide a reasonable but limited set of functionality.

3.4 Support for satellite portals

In many emerging distributed grid infrastructures, it is desirable to have multiple different portals all running with the same authentication information. In GEON the impetus for this is to enable the partner institutions to develop their own GEON applications in their own portal customized for their institution. While the portal applications are left to the institutions to develop, each of these satellite portals should be hooked into the global GEON services such as security and data management. In this section, we describe the features of GAMA that help to facilitate these satellite portals.

To keep control of the identity of the users, GEON must ensure a consistent account acceptance policy. To do so, we mandate that a single portal instance be used for account requests and acceptance. This portal must have increased security in order to communicate with the secure port of the GAMA server, and will define the policies on which users get accounts. The satellite portals can be configured to point to this special authoritative portal for account requests.

Once the users are defined, the satellite portals have the ability to "import" account information directly from the GAMA server. Figure 3 shows the interface for this. In this example, the portlet shows six accounts that exist on the GAMA server that can be imported. The administrator checks the accounts that should be imported and the system imports all the information needed. After import, satellite portal logins for imported users are processed by the GAMA server as described above in the User Login Process section.


Account Import M

Security server users

Account(s) successfully imported

Refresh user list Add selected users to portal

These users exist on the security server but not on this portal:

Select	User Name	
✓	testing123@sdsc.edu	/C=US/O=geongrid/OU=sdsc/CN=realt
☐	cell@sdsc.edu	/C=US/O=geongrid/OU=sdsc/CN=Cell
✓	tester3@test.com	/C=US/O=geongrid/OU=sdsc/CN=test
✓	gar@sdsc.edu	/C=US/O=geongrid/OU=sdsc/CN=Gar
☐	test6@sdsc.edu	/C=US/O=geongrid/OU=sdsc/CN=test

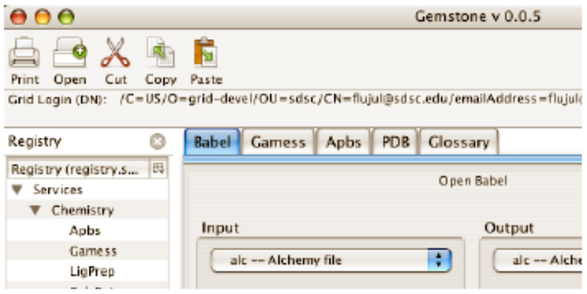
Figure 3: Administrative interface for "importing" grid accounts.

4. Other clients

While portals provide the primary interface to the GAMA server for the user and the administrator, the architecture supports additional client tools including rich clients and command-line executables.

Rich clients such as Java Web Start applications and Flash applications offer additional flexibility over the interfaces that can be provided in web portals; for example, rich clients provide more interactivity and better visualization interfaces than are typically provided through the browser.

Once a user account is created on the GAMA Server using the portal-based interfaces, the user can access a rich client that has been configured to work with GAMA. One such client is the Gemstone client being developed in the domain of computational chemistry. The Gemstone interface contacts the GAMA server using standard web services (SOAP) protocols and retrieves a grid proxy credential on behalf of the user. Figure 4 shows a portion of the Gemstone interface and the DN of the user logged in.



Grid Login (DN): /C=US/O=grid-devel/OU=sdsc/CN=flujul@sdsc.edu/emailAddress=flujul

Figure 4: GEMSTONE uses the GAMA Server to authenticate users with web service APIs.

We have recently started work on a set of command-line tools that will simplify the process of creating and administering user accounts on computational clusters using GAMA. Information retrieved from a GAMA Server may be used to create standard unix accounts on cluster front-ends and configure the cluster grid environment for new users. For instance, some cluster administrators follow a certain set of steps when adding a new user to their clusters, such as 1) add a new unix user, 2) add an entry in the Globus grid-mapfile for the new user with the user's unique DN, and 3) add an authorized_keys file to the user's .ssh directory to allow passwordless ssh access by the user. Since we already store username and DN information in the GAMA server, we can automate steps 1 and 2 at the present time through the gamacl (GAMA command-line) client which is called as follows:

```
   % java -jar gamacl.jar useradd
username -gamaserver hostname:port -
localname localusername -addmap -
mapfile mapfilename
```

This command will retrieve information about the user specified by *username* from the GAMA Server specified by *hostname:port* and add a new local unix user with *localusername* and add an entry for this user to the grid-mapfile specified by *mapfilename*. The GAMA command-line tools are in the early stages of development, and we only show this particular command to provide an example of the types of functions we think would be useful to system administrators. Many more useful command-line features will be possible once we have implemented some of the architectural changes to the GAMA Server outlined in the following section.

5. Evaluation and discussion

GAMA was designed to simplify the deployment and configuration of distributed grid systems such as GEON and Telescience. Many emerging grid systems follow a similar architecture and use similar tools, and therefore, we believe that GAMA can easily be deployed for these other projects. GAMA leverages existing, trusted software components such as CACL and MyProxy and provides backward compatibility with legacy applications such as GSI-enabled SSH as well as newer rich-client applications currently being developed.

From an end-user perspective, the GAMA interface, modeled after common practice within the commercial web development domain, seems to provide the users with the appropriate level of abstraction with which they are comfortable while providing the strong security guarantees that system architects desire.

The performance of the web services wrappers on the server side to date has been acceptable. Initial benchmarks show that login times are in the range of 2-3 seconds even across the wide-area network; account creation takes longer, in the range of 6 seconds, mostly due to the creation of the certificate on the server; and account imports are under a second.

While we believe that GAMA is suitable for creation of new grid systems, it is less suitable for grid systems already extant. If user certificates have already been issued, they may not be easily imported into the GAMA Server. In addition, some grid systems use a federated trust model where multiple certificate authorities exist at different institutions all of whom agree to accept each others' certificates. GAMA currently has no way to accommodate such an architecture.

The next release will also include better user information management on the server, automatic synchronization between clients and server, and additional interfaces for administrators to define which accounts to synchronize between which resources. The CAS integration, while functional in our prototypes, is not packaged as part of the version 1.0 release but will be packaged in a subsequent version.

6. Related work

6.1 PURSE

The PURSE project [5] is, on the surface, quite similar to GAMA in that it bundles a CA system and a MyProxy credential repository on a dedicated server and provides a portal interface for account request creation and management. However, the PURSE portal actually runs on the same machine as the CA and MyProxy packages, in contrast to the GAMA system in which account request and administrative portlets are installed on a remote portal that communicates with the GAMA server via web services. The end result of a new user being created through PURSE is that the user's GSI credentials exist in a MyProxy repository on the PURSE server. In order to be of use, a proxy must be retrieved from the MyProxy server by a portal or other environment, and this step is outside of the scope of PURSE. The benefit of having GAMA portlets on a remote portal is that we actually provide the proxy retrieval step, whereby a proxy is automatically and transparently retrieved from the GAMA server and activated in a portal environment when a user logs in to the portal. This can make application development easier for portal developers because they have access to a user's proxy "for free" in the portal environment, and they do not have to know anything about MyProxy or GAMA. At the same time, a proxy can also be retrieved directly from the GAMA MyProxy server by other mechanisms such as would be used in a PURSE rollout. In addition, GAMA Server functions are exposed as web services so they can be accessed from multiple portals and other environments, such as rich client applications.

PURSE does not explicitly manage user authorization. Authorization happens downstream through standard GSI mechanisms at the resource level when a user who has retrieved a proxy from the PURSE MyProxy server passes that proxy to an end resource and attempts to perform some operation. It is up to the end resource to determine if the user is authorized to perform that operation based on internal policies. GAMA also relies on such mechanisms when it is configured to use CACL and MyProxy, though it does have an option to use CAS certificates which explicitly add authorization information to the retrieved proxy. However, CAS support is not entirely stable nor especially useful at the moment because not many end resources know how to use CAS authorization assertions.

6.2 GridAuth

GridAuth [23] is comprised of a server component, written in Perl, and a set of client implementations (currently Perl, Java and PHP are available). GridAuth is based on a configurable plug-in architecture whereby certain pre-defined operations (such as login, adduser, etc.) are exposed through a standard API and may be implemented on the server via various mechanisms. GridAuth comes with a custom CA package for credential creation and management, and enables credential retrieval upon user login. Prospective users can request an account through a simple web interface on the server, and administrators manage account requests through a similar interface. The set of steps that are taken when an account request is approved (for instance) depends on the specific plug-ins that are in place on the server and are configured to handle the useradd() function. This method of allowing and managing unique implementations of standard API-specified functions is quite powerful, and certainly provides more flexibility in this regard than GAMA which is coupled rather tightly to CACL and MyProxy in the current version despite the fact that we expose implementation-independent methods such as login() through web services. GAMA will be more implementation-neutral in the future, though it will still come with MyProxy and CACL options in addition to others.

GridAuth comes with client implementations in various languages, as already mentioned, which must be integrated into some other application or portal environment in order to be useful. As in the previous comparison with PURSE, GAMA has an advantage here in that we have already integrated it into an application development environment (GridSphere) to minimize the workload and knowledge required of developers.

The GridAuth API does not explicitly support authorization, but instead relies on GSI mechanisms at the resource level. It is possible that authorization assertions could be stored in and retrieved through GridAuth, though no standard way of doing so is specified or supported out of the box.

6.3 DOE FusionGrid

The DOE FusionGrid [24] project has been evaluating and using various user management and security systems for some time. Most recently, they have created a system that, like GAMA, uses a dedicated server with a CA system and MyProxy installed on it. Like PURSE, there is a web interface directly on the server for handling user account requests and server management. The CA software was written in-house due to security requirements of the

FusionGrid collaboration. Clients retrieve credentials directly from the MyProxy server using MyProxy client tools, as in PURSE, so client applications and/or users must be MyProxy-aware. Again, this is in contrast to GAMA where credential retrieval in a portal environment is hidden from users and application developers completely.

A distinct advantage of the FusionGrid system over GAMA, PURSE and GridAuth is its authorization system called ROAM. Where the others mostly leave authorization to end resources (typically through the rather coarse-grained and decentralized grid-mapfile mechanism), ROAM provides application developers and system administrators with an easy-to-manage and comprehensive system for ensuring that users are allowed to access resources in the FusionGrid. The CAS package in GAMA is intended to solve many of the same problems, and in a more standards-compliant way, but its true usefulness has yet to be demonstrated.

While GAMA, PURSE and GridAuth are packaged and distributed freely, it appears that the FusionGrid solution is proprietary. Because it evolved in a very specific environment to meet the needs of a certain set of users, it is no surprise that the end result appears to be rather tightly coupled to its environment and user base. Though GAMA and GridAuth (and perhaps PURSE) were also developed to meet the needs of a specific community, the developers have put considerable effort into the task of making software that can be shared and used in other communities.

Acknowledgements

Supported in part by NBCR P41 RR08605. Contributions were made to this work by GEON, an NSF funded Collaborative ITR Project.

7. References

[1] Foster, I., C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of High Performance Computing Applications*, 2001. **15**(3): p. 200-222.
[2] Foster, I., C. Kesselman, G. Tsudik, and S. Tuecke, "A Security Architecture for Computational Grids", *ACM Conference on Computers and Security*. 1998. p. 83-91.
[3] Tuecke, S., et al., "Internet X.509 Public Key Infrastructure Proxy Certificate Profile", 2003, IETF.
[4] Mock, S., et al., "The GridPort Open Source Portal Toolkit", http://gridport.sourceforge.net/
[5] GridCenter, N., "A Portal-based User Registration Service for Grids", April, 2005, http://www.grids-center.org/solutions/purse/
[6] Thomas, M., et al. "The Gridport Toolkit: a System for Building Grid Portals", in *10th IEEE International Symp. on High Perf. Comp*. 2001.
[7] Link, W., "CACL, A CA System with Automated User Authentication", Sept 2003, San Diego Supercomputer Center, http://www.npaci.edu/CA/cacl.pdf
[8] Welch, V., et al. "Security for Grid services", in *Proceedings 12th IEEE International Symposium on High Performance Distributed Computing*. 2003.
[9] "MyProxy Online Credential Repository", http://grid.ncsa.uiuc.edu/myproxy/
[10] Novotny, J., S. Tuecke, and V. Welch. "An Online Credential Repository for the Grid: MyProxy", in *High Performance Distributed Computing (HPDC)*. 2001.
[11] Thompson, M., et al., "Certificate-based Access Control for Widely Distributed Resources", in *Proc. 8th Usenix Security Symposium*. 1999.
[12] Hallam-Baker, P. and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)", May 2002, OASIS, Committee Specification, 01, http://www.oasis-open.org/committees/security/docs/cs-sstc-core-01.pdf
[13] OASIS, "Security Assertion Markup Language (SAML): Assertions and Protocols", Aug 2003, OASIS, oasis-sstc-saml-core-1.1, http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf
[14] Pearlman, L., et al. "A Community Authorization Service for Group Collaboration", in *IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*. 2002.
[15] Pearlman, L., et al. "The Community Authorization Service: Status and Future", in *CHEP*. 2003.
[16] Alfieri, A., et al. "VOMS, an Authorization System for Virtual Organizations", in *1st European Across Grids Conference*. 2003. Santiago de Compostela.
[17] Baru, C. and others, "CyberInfrastructure for the Geosciences", http://www.geongrid.org
[18] "Telescience: A Collaborative Environment for Telemicroscopy and Remote Science", 2004, https://telescience.ucsd.edu/.
[19] Bruno, G., M. Katz, F. Sacerdoti, and P. Papadopoulos, "Modifying a Standard System Installer to Support User-Customizable Cluster Frontend Appliances", in *IEEE International Conference on Cluster Computing*. 2004: San Diego, CA.
[20] Papadopoulos, P., M. Katz, and G. Bruno, "NPACI Rocks: Tools and Techniques for Easily Deploying Manageable Linux Clusters", in *Concurrency and Computation: Practice and Experience*, Special Issue: Cluster 2001, 2002.
[21] "Portlet Specification", 2003, The Java Community Process, JSR 168, http://www.jcp.org/aboutJava/communityprocess/review/jsr168/
[22] "GridSphere Portal Framework", http://www.gridsphere.org/
[23] "GridAuth", http://www.gridauth.com
[24] Burruss, J.R., Fredian, T.W., Thompson, M.R., "Simplifying FusionGrid Security", Challenges of Large Applications in Distributed Environments (CLADE) workshop at HPDC14, July 2005