Review of Security Models Applied to Distributed Data Access

Antonia Ghiselli[2], Federico Stagni[1], and Riccardo Zappi[2]

[1] Istituto Nazionale di Fisica Nucleare sez. di Ferrara,
via Saragat 1 - 44100 Ferrara, Italy
{federico.stagni}@fe.infn.it
http://www.fe.infn.it
[2] Istituto Nazionale di Fisica Nucleare CNAF,
viale Berti Pichat, 6/2 - 40127 Bologna, Italy
{antonia.ghiselli,riccardo.zappi}@cnaf.infn.it
http://www.cnaf.infn.it

Abstract. In this paper, we explore the technologies behind the security models applied to distributed data access in a Grid environment. Our goal is to study a security model allowing data integrity, confidentiality, authentication and authorization for VO users. We split the process for data access in three levels: Grid authentication, Grid authorization, local enforcement. For each level, we introduce at least one possible technological solution. Finally, we show our vision of a SOA oriented security framework.

This work is developed as part of the CoreGRID Network of Excellence, for the Institute on Knowledge and Data Management.

Keywords: Grid, data management, security, authentication, authorization, policy, acl, XACML, SAML.

Introduction

In this report, we will explore the technologies behind the security models applied to distributed data access in a Grid environment. Our goal is to study a security model allowing data integrity, confidentiality, authentication and authorization for VO (Virtual Organizations) users [13]. Although the effort will be to create a generic model, the work will be based on a Grid framework with the following assumptions: Grid users are organized in VOs with existing tools to manage memberships and credentials. In other words, we want to define policies for resource usage on the basis of user credentials, and to enforce them on the basis of Grid status.

The rest of this paper is organized as follows: in section 1 we introduce our approach to security with some definitions. In section 2 we explain some general requirements. In section 3 we describe the technologies to build a security framework. In section 4 we introduce the technologies to build a Grid data access framework.

W. Lehner et al. (Eds.): Euro-Par 2006 Workshops, LNCS 4375, pp. 34–48, 2007.

1 Definitions

Initially, the Grid was referred to as *Computational Grid*, thinking as a way to share computational facilities. However, much of the Grid jobs are data intensive, and to stress this point, today we normally think of Grids in term of *Data Grids*: most large jobs that require Grid services, especially in the scientific domain, involve the generation of large datasets, and their consuption [1]. There is a necessity for the reservation and the scheduling of data repositories, and so we need to express some policies to govern their access. Moreover, thousands of people may want to use storage resources to share him/her data with a limited set of other researchers, or maybe with no one but themself. The future storage systems will contain critical user information for various applications and purposes, like for example life science and financial ones.

Grids need an authorization framework to handle the users privacy necessities, and their limits too. In other words, we want to control the access to the Grid users' data on the basis of some high controlled sharing rules.

1.1 Grid Data Management Systems

A distributed system is a collection of independent computers that appears to its users as a single coherent system. Similarly, a distributed data access system is a distributed data storage, with ubiquitous and transparent data access and migration. A *Grid Data Management System* (GDMS) is a data access system acting in a Grid environment. GDMS offer a common view of storage resources distributed over several administrative domains. Therefore, they must allow the smooth integration or removal of resources, without affecting the integrity of neither the individual independent domains nor the system as a whole. Problems behind the implementations of such a system are:

- *processes communication:* the way distributed processes exchange informations.
- *Naming:* name resolution and localization.
- *Synchronization, consistency and replication:* the way data are synchronized and the definition of policies for the consistency and the replication of data.
- *Security:* the way to gain security for data access.

1.2 Security

We define a security architecture as *a set of features and services that tackles a set of security requirements and can handle a set of cases*[16]. Grid systems in use today do not address security in a systematic way: just to make an example, historically in Globus [17] an authenticated user is a good user. This emphasizes the authentication aspect, but Grids need a strong authorization mechanism. Our aim is to enable new Grid infrastructure developer to create more secure systems, capable to attract new Grid users and applications. Security models should define "who can do what, when and where". A Grid middleware should

encompass a security framework, in which we can distinguish two virtual black boxes: the *authentication* box and the *authorization* box:

- *authentication* deals with the verification of the identity of an entity within a network. An implementation should provide an agnostic plug point for multiple authentication mechanisms, and the means for conveying the specific mechanism used in any given authentication operation.
- *Authorization* deals with the verification of an action that an entity can perform after authentication was performed successfully. The goal of an authorization framework is to provide a light-weight, configurable, and easily deployable policy-engine-chaining infrastructure that is agnostic to back-end enforcers and evaluators, as well as the run-time container infrastructure and the state model that hosts them. The framework allows for a combined and flexible decision making process, taking into account information, assertions and policies from a variety of authorities.

We can make a brief comparison between the high-level techniques besides authentication and authorization. The first link in the Grid security chain is authentication. Grid resources authenticate remote users using basically two ways: the first uses a session key, and the second, which is the mostly used too, uses the Public Key Interface (PKI). On the other hand, we need a Privilege Management Infrastructure (PMI): a PMI is to authorization what a PKI is to authentication [2]. Just to make an example, we can express some user's attributes using the X.509 Attribute Certificate (AC), which maintains a strong binding between a user's name and its attributes. Certification Authorities (CAs) digitally sign a public key certificate; in a similar way, the entity that signs an AC is called an Attribute Authority (AA), while the root of trust of the PMI is called the Source of Authority (SOA), which may delegate its power to subordinate AAs. Like Certificate Revocation List (CRL), an AA could issue an Attribute Certificate Revocation List (ACRL) to revoke privileges from an AC. Obviously, ACs is just one of the possible solutions to join users and their attributes.

2 Requirements

Integration, interoperability and trust are the building blocks of the requirements behind a Grid security infrastructure. In this section we give some brief and general guidelines, but we want to point out that more specific requirements will be glean in the proceeding of this paper.

- *Confidentiality* is the property that information doesn't reach unauthorized individuals, entities, or processes. It is achievable by a mechanism for ensuring that only those entitled to see information or data can access them.
- *Integrity* is the assurance that information can only be accessed or modified by those authorized to do so. Data integrity is a nontrivial problem especially when storage hardware and networks are not perfect.

- *Resilience* is an important requirement as the Grid links and nodes are very dynamic in nature and may change over the time. The GDMS security architecture should remain intact and should deliver the promised level of security assurances even if its composition changes over the time. The resilience provides an abstraction layer to hide the architectural changes from the overall security architecture.
- *Data Lifecycle Management (DLM)* is the process of managing data throughout its lifecycle. GDMS should ensure that the data contents will be protected from malevolent entities.
- *Fault Tolerance* is a desirable feature especially when transfers of large data files occur.

2.1 Data Types

In section 1, we've made a really brief history of the evolving of the concept of Grid, from Computing Grids to Data Grids, and we mentioned the data types involved in it. To understand all the security requirements, we have to think to who is using Grid now, who is going to use it soon, and who wish to use it, but can't trust it for some security reason. At the present time, the majority of Grid tools are growing behind some specific needs, mainly HEP[1] experiments. These applications produce and consume a considerably high amount of data with heavy impact on the bandwith, but probably they don't need a high security system, because the main purpose of this activities is to be fast. In the future, much more people is going to use Grids and peer-to-peer systems, not only with the actual purposes. In the next generation file sharing, a user will want to give access to his/her files only to a limited set of people. There's the need for a high control over who is authorized to view them. This means more protection levels, but less performance too. What we want to stress here, is that every data type needs different protection levels, and that a Grid security system must take care of this principle. Different data types can determine the way we achieve data integrity, confidentiality, authentication and authorization.

The next generation storage elements would be able to publish the Quality of Protection (QoP) they can assure to the data they own. In this way, the QoP will be decisive for the entire data storage system. Just to make an example, a user should request the resource provider to not have read access to his/her data: this is a non-trivial challenge, and obviously not all the storage elements will be able to enforce this demand, because this is depending of the locally implemented security system: it determines the QoP and the Service Level Agreement (SLA), which defines how data is protected while in transit over the service. Security negotiations should be used to establish secure sessions between the endpoints. A security infrastructure featuring support for negotiations and establishment of end-to-end and/or hop-to-hop security associations has broader applicability to general networked environments like Grids. Security negotiations require some brokering agent to mediate between the endpoints.

[1] High Energy Physics.

3 Data Management Security Technologies

We can roughly divide the process to reach access to a Grid resource in 3 levels: first of all there's a Grid authentication process, then authorization on a Grid-ID base, and finally local enforcement using the resource-specific security framework. In figure 1 you can see the all-round security process.

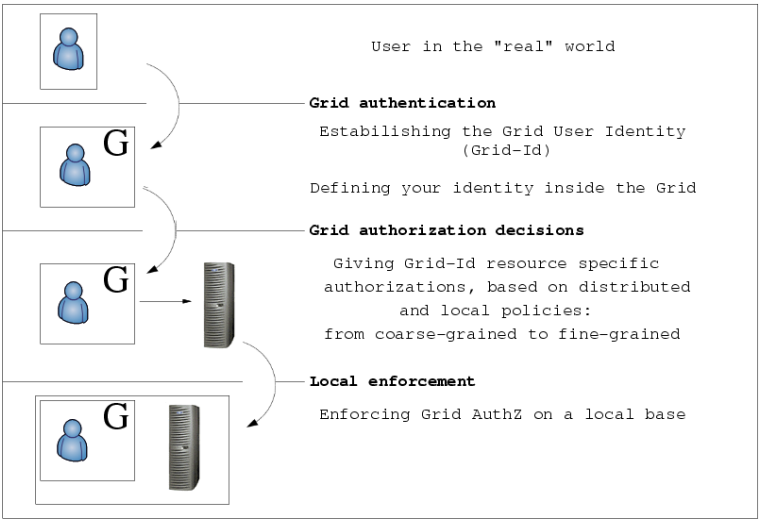

Fig. 1. The security process

3.1 Authentication

Grid computing is, in its essence, about bridging organizational boundaries. In order to do so, we can report here two commonly identified solutions: *virtual organizations* [13] and *federated trust*.[2] We are not going to explain the difference between these two models, because they are quite theoretical and in practice it is often hard to distinguish the boundaries between them. Grid users are traditionally organized in VOs.

In a Grid environment the authentication model is normally based on the concept of *trusted third parties* (TTPs): the first link in the authentication chain is the certification authorities (CAs), which in practice are trust anchors for VOs. This model makes use of the Public Key Infrastructure (PKI) technology: CAs issue X.509 certificates, where essentially a unique identity name and the public key of an entity are bound through the digital signature of that CA. It is possible that some GDMS may require further security controls, but these issues are out of the scope of a Grid authentication service, because they suppose a specific contract between the user and the resource, outside the Grid security infrastructure.

[2] For more information, see http://www.projectliberty.org/

An authentication service must define distinctly the Grid identity of any user: this mean that every user inside a Grid is given a background, a description. With description we mean not only user's VOs, but his/her role inside every VO he is member of. In the proceeding of this paper we will refer to this kind of enhanced authentication as of "Grid authentication".

Role Based Access Control. Access Control technologies has evolved from two fundamental types: Discretionary Access Control (DAC), and Mandatory Access Control (MAC). DAC permits the granting and revoking of access control privileges to be left to the discretion of end users, typically the resource owner. MAC is a way of restricting access to objects based on the sensitivity of the information contained in the objects. These policies aren't well suited for VOs authorization requirements, because we need to take access decisions on the basis of the roles that individual users have as part of an (Virtual) Organization. In the Role Based Access Control (RBAC) [5] user access rights are defined by roles in the form of user attributes, letting a separated management access control policy defining what roles are allowed to do what actions on resources. The roles represent typically organizational roles such as secretary, manager, employee, etc. In the authorization policy, they are given a set of permissions, and each user is then assigned to some or more roles. When accessing a target, a user establishes a session and, during it, he can request the activation of some of the roles he is authorized to play. After that, the user will be represented by his/her roles, and so the authorization framework will deal with roles rather users themselves.

We present here two existing estensions. The first is the *hierarchical* RBAC model, which is just a more sophisticated RBAC type, in which the senior roles inherit the privileges of the more junior roles. For example, there might be the following hierarchy:

$$\text{employee} \leq \text{programmer} \leq \text{manager} \leq \text{director}$$

Giving the role "programmer" some permissions means that managers and directors will inherit them. The hierarchical extension to RBAC fits very well the Grid VO requirements, and so we assume that the Grid end systems, like the storage ones, will be able to enforce the capabilities applied to VO roles in a hierarchical fashion.

The second RBAC extension is the *temporal* RBAC model (TRBAC) [6], which supports periodic role enabling and disabling, and temporal dependencies among such actions. Consider for example the case of a part-time staff in a company: what we want to do, is to give him authorization only on working days. With TRBAC, we can assign the part-time staff a role, and enable it only during a temporal interval. The role enabling/disabling depends on some requirements, that can be used to constrain the set of roles that a particular user can activate at a given time. Enabling/disabling actions can be given a priority to help in solving conflicts.

An RBAC system will become a must to manage the future Grid authorizations. Without it, the wide mutable nature of VO-like systems would become a

nightmare for all systems administrators, who should take care of granting every single user with his/her capabilities. RBAC simplify the VO's administrator life too, because they have just to assign every user with a somewhat restricted set of roles. In this way the user's identity is managed at VO-level, while the end-systems deal with roles only. We can have the right granularity level with the less possible effort.

Anyway, this isn't perfect yet: assume that a user, like

$$User = John_Doe/VO = NeVO/Roles = ExRole, NeRole$$

is trying to do something nasty, for example he is using his *ExRole* capabilities to store a malware in his role shared space. If the system (or the administrator) recognizes it, it should be possible for him to boot that user from his resources without affecting his entire VO/role. There are two possibilities:

- if the end-system doesn't deal with the users authentication names, the only possibility is to do a report to the *NeVO* VO, asking it to reject that user: to do this, there is first the need to recognize the user, and this isn't practical nor fast.
- The second and best option is to let the end-systems to know the effective user names, although the policy end-systems should only use the VO/roles associations to determine the capabilities. In other words, the end-systems authorization frameworks should use the effective user names only for in-depth security reasons.

State of the Art: VOMS. In 3.1 we have stated that, from our point of view, a "Grid authentication" should give a user a complete background and identification. Actually, a framework that can be used to reach this objective is the Virtual Organization Membership Service [8], which is an accepted authentication and authorization framework into existing Grid projects, like for example EGEE[3] and OSG.[4]

VOMS is an Attribute Authority (AA). Users can be organized in a hierarchical structure with groups and subgroups, thus implementing a hierarchical RBAC system. To allow for more flexibility, users are also characterized by two other sets of credentials: roles and capabilities. Roles are used to specify the users' properties as members of some groups. The main difference between groups and roles is that a user can choose which of his roles are to be listed in his credentials, while all his groups are always specified. Capabilities are expressed as free-form strings of characters, and can be used to describe the user's special characteristics. VOMS is traditionally presented as an authorization framework, but in this paper we introduced it as an authentication one. The reason besides this choice is that VOMS is used to define the "Grid user identity" (it can provide a "Grid authentication"), which is not a grant for authorization on any end system: the enforcement of these VO-managed attributes at local level must reflect

[3] http://public.eu-egee.org/
[4] http://www.opensciencegrid.org/

the agreements between the VO and the Resource Provider (RP). However it should be possible for an RP to override the permissions granted by a VO, for example banning unwanted users.

3.2 Authorization

Authentication frameworks and Attribute Authorities (AA) can provide a coarse-grained granularity to identify the users' roles and background in a Grid. An authorization system can make use of these information for fine-grained access decisions, using a policy authorization service.

Evaluating the Policies. In this section we will explore the policy interactions and their relations with the Privilege Management Infrastructure (PMI) introduced in section 1.2. First of all, we have to remind some of the requirements for a policy authorization service:

- a future authorization service will be based on a recognized policy expression language and exchange format, and will use a Request/Response protocol to allow intra-site and multiple site scalability. This implies the investigation for the use of "standard" format languages and protocols.
- It will use the principle of ownership in respect to the policy and decision making precedence. This means that the final decision will always reside with the resource owner. It should be able to explicitly accept or reject policies from other domains, and to distribute them.
- A future authorization service will separate authorization infrastructure from the policy itself, providing only secure environment and mechanism for site-authority controlled policy enforcement. The policy evaluation engine will be implemented as a separate service that will be able to call external separate decision points.

We can give a more formal specification for this requirements, using the following definitions:

Policy: The combination of rules and services, where rules define the criteria for resource access and usage.

Policy Decision Point (PDP): The point where decisions about the policies are made. It evaluates *applicable policies* and renders *authorization decisions*. In a loosely coupled distributed environment like Grid, a local to a resource (designated) PDP can call other PDPs requesting for evaluating policy components related to their domain of authority to provide a final decision.

Policy Enforcement Point (PEP): The point where the policy decisions are actually enforced. This is the system entity that performs *access control*, by making *decision requests* and enforcing *authorization decisions*. From a data management perspective, this means that every storage resource should have a local PEP to enforce the policy decisions.

Policy Authority Point (PAP): The point that owns the authority over the PDPs. We should remind that sometimes PAP indicates the *Policy Administration Point*, which is the system entity that creates and administer the policies.

Policy Information Point (PIP): The system entity that act as a source of attribute values.

The PDP-PEP interaction is the key for a good policy distribution. There are two possible basic implementations, the pull model and the push model. The pull model is the more used one, in which a supplicant first ask for the resource PEP to authorize himself, and then the PEP ask to an external PDP for the final decision. We can see a brief example to clarify the way these policy points interact each other. To allow user access on a storage resource, for example a SRM[1] implementation, the storage agent requests via his own PEP an authorization decision from a designated PDP, that evaluates the authorization request against the policy defined for the request, resource and user attributes/roles. During the policy evaluation, the PDP may also request specific user attributes from a Policy Information Point (PIP), or asking an authentication service for user identity confirmation. It should be noted that these controls are a burden for a high percentage of the actual Grid data, but should be a must for some of the future Grid storage uses: data owners and the system administrators should be able to choose how much security controls will be needed. When the PDP identifies the applicable policy instance, it collects the required context information, evaluates the request against the policy, and communicate the decision back to the PEP. After receiving a PDP decision, the PEP conveys the service request to the resource, that may also have a locally determined policy implying additional restrictions on resource usage and/or access. All these communications can be secured using cryptographic technologies like SSL/TLS or MLS.

In essence, when making an authorization decision, we should be able to combine information from a number of different sources. In other words, policies should be defined at different levels, like VO, site, or other stakeholders. It should be noted that every level should have the permission to define different kind of policies, and that sometimes they could overlap each other. For example, a VO-level PDP could force some of his self-managed group/role to not exceed a disk quota of 100 Mbytes, but a resource-level PDP should impose a more restrictive permission. We think that every controversial decision should be resolved in favor of the local decision point, which could be the PEP closest match. This PDP, called "Master PDP", composes the final decision, optionally contacting other PDPs. In figure 2, we show a possible interaction flow between Policy Points.

The solution presented above has known performance problems: requesting a remote PDP decision involves the use of time and resource hungry components, such as building a remote SSL/TLS connection, message parsing, possible remote policy request and PDP/AuthZ service invocation [11]. For this reasons, there's the need for investigation over the PDP-network topology, in order to avoid useless communications. This trade-off can be resolved using distributed policy caching, combining pull and push operation models, using short-validity authorization tickets, or implementing a policy guessing mechanism. In addition,

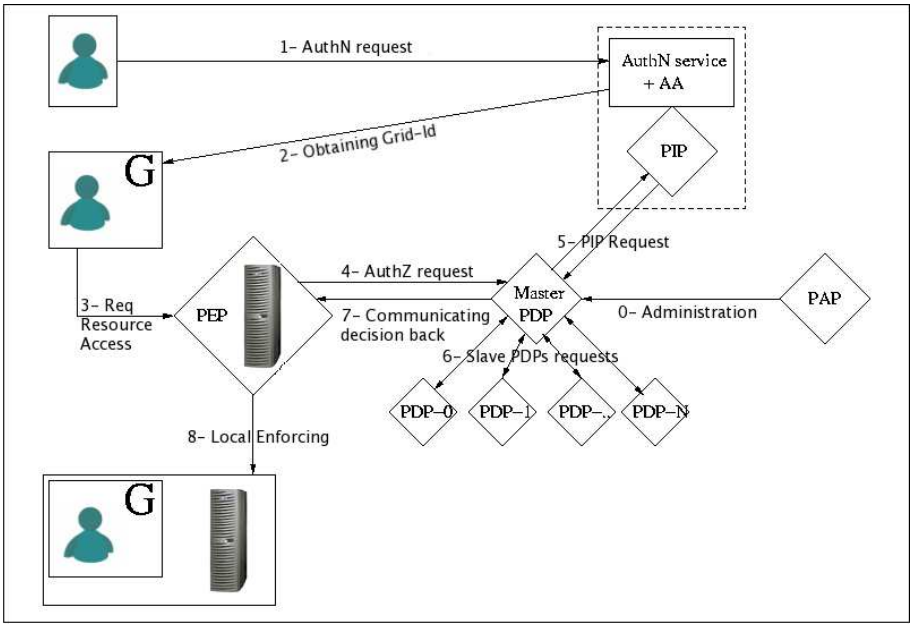

Fig. 2. Interactions between Policy Points in an all-round authorization process. Not all of the shown communication are mandatory.

everyone of the listed services should be a bottleneck for the entire authorization framework; in this situation, consider for example the devastating effects of a Denial Of Service (DOS) attack to anyone of the listed policy points.

Using a standard policy language. Nowadays, the language for writing access control polices that best fit the listed requirements is the eXtensible Access Control Markup Language (XACML) [20], which is an XML based technology developed and standardized by Organization for the Advancement of Structured Information Standards (OASIS).[5] It should be considered a "de facto" standard for expressing policies. XACML includes an access control language, a processing environment and a request-and-response protocol that let developers write policies that determine what users can access on a network or over the Web. XACML can also be used to connect disparate access control policy engines. Every policy is defined for the target triad "Subject-Resource-Action". The processing environment assumes interactions between the policy points described in the previous section.

XACML has many benefits over other access control policy languages:

– One standard access control policy language can replace dozens of application-specific languages.

[5] http://www.oasis-open.org/

- Administrators save time and money because they don't need to rewrite their policies in many different languages.
- Developers save time and money because they don't have to invent new policy languages and write code to support them. They can reuse existing code.
- Good tools for writing and managing XACML policies will be developed, since they can be used with many applications.
- XACML is flexible enough to accommodate most access control policy needs and extensible so that new requirements can be supported.
- One XACML policy can cover many resources. This helps avoid inconsistent policies on different resources.
- XACML allows one policy to refer to another. This is important for large organizations. For instance, a site-specific policy may refer to a company-wide policy and a country-specific policy.
- It provides facilities to support the core and hierarchical RBAC approach.

Anyway, XACML doesn't define protocols or transport mechanisms to protect the message security with authenticity, integrity and confidentiality. Full implementation of this model depends on use of other standards, for example the OASIS Security Assertion Markup Language (SAML) [19] [21]. SAML is an XML standard that supports web single sign on, attribute-based authorization and securing web services. There are threes basic SAML components: assertions, protocol, and binding. Assertions can be one of three types: authentication, attribute, and authorization. Authentication assertion validates the identity of the user. The attribute assertion contains specific information about the user, while the authorization assertion identifies what the user is authorized to do. The protocol defines how SAML request and receives assertions. There are several available binding for SAML, that define how message exchanges are mapped to SOAP, HTTP, SMTP and FTP among others.

State of the Art: Gridmap File, CAS, G-PBox. One of the first attempt to provide authorization in Grid was in the form of the Globus Gridmap File. This file was simply a list of the authorized user, identified by a distinguished name, and the equivalent local user account name they are to be mapped into. This solution is infeasible for the next generation Grids, because the resource owner can't set a policy for who is allowed to do what, and maximize the workload of the resource administrator who must keep track of all the authorized users. This system isn't scalable nor flexible [3].

The Globus team developed the Community Authorization Service (CAS) [9]. CAS allows for a separation of concerns between site policies and VO policies. It allows the resource owner to grant access to a portion of his/her resource to a VO. The CAS server acts as a trusted intermediary between VO users and resources: the users first contact the CAS asking for a permission to use a resource, the CAS server consults its policy, and grants or deny the access. CAS does not issue Attribute Certificate's (AC), but whole new proxy certificates, and this isn't a good solutions, because a security system should use standards. Another

problem is that CAS completely remove control from site administrators, and that it requires a VO to know everything about the configuration of its farms.

One of the most interesting authorization framework is the Grid Policy Box (G-PBox),[7] which can be used for the representation and management of policies for Grid infrastructures. It's based upon the composition of modular objects, Policy Boxes (PBox), which are policy repositories hierarchically-distributed to independent administrative-based layers, each containing only policies regarding itself. In G-PBox, there are PBoxes at VO, domain, farm and site level, with the possibility to have sub-farm levels. Each and every client that wants to be policy-aware, has a configured PBox that will be contacted whenever a policy decision is required. From a theoretical point of view, we can think at every PBox as being a Policy Authority Point containing a Policy Decision Point, while every resource should have a local Policy Enforcement Point. In G-PBox, the policies are defined using XACML.

3.3 Local Security Enforcement

There are basically two ways to enforce access control over data: the first is to allow Grid access only, the second is to allow local access in parallel to Grid access.

In the *Grid enforced security model*, users can access their files only via Grid tools and services. As stated in [14], "the easiest way of implementing this is to assign all files in a storage element to a service userid, for example *gstorage*, and to add a component, which runs under this identity and interacts with the user": the users should go through the Grid middleware services to gain access to their files. With this type of service, we can easily have a standard authorization service for all the Grid resources, with uniform security semantics, that can take authorization decisions like a centralized authorization service. This model gives support for resources with weak local authorization mechanism.

In the *Site enforced security model*, if we want to allow local access in parallel to Grid access, we have to implement a mapping from Grid identities to local userids. If a Grid service has to act on the user's behalf, then it needs the user's credential to be delegated. With this model, the site storage administrator has full control over his resources, and he will be able to use the local authorization mechanism he prefers.

The choice from a Grid user's point of view should be the Grid security model, because it integrates the site peculiarities into a uniform security model, where every Grid storage site looks the same. Although this last point could be reached also in the site enforced security model using an additional layer for the standardization, we have to remind that an external security service will let site administrators to administer their own local security, in a site technology independent fashion. The real problem besides this is that the Grid security model is not acceptable by some sites due to their local policies, and in addition existing security infrastructure can't be replaced overnight. Each domain typically has its own authentication and authorization infrastructure that is reputed secure and reliable, and site administrators won't replace it in favor of a single new

model or mechanism. From the beginning of the Grid computing one of the fundamental requirements was to let every site to use his own security mechanism, and this implies the use of a site enforced security model. In the Globus Toolkit [17], gateways are used to translate between the common GSI infrastructure and local site mechanisms, for example Kerberos Identities or local UNIX users and groups. In LCMAPS [18], we can map Grid users to local ones, and primary and secondary local groups, which are predefined by the resource owner: LCMAPS is used to delegate some global Grid credentials to the local site security system, in this case the UNIX uid/gid match, with the possibility to add ACLs if the file system (and the kernel) can handle them.

Access Control List. Access Control List (ACL) is a means of determining the appropriate access rights to a given object, depending on certain aspects of the process that is making the request, principally the process's user identity. This is a deliberately general definition, because ACLs have been implemented in many ways in different environments.

The POSIX.1e ACLs [12] are an estension of the POSIX.1 permission model, the standard 9-bit access permissions of the UNIX systems. The extended ACLs support more fine-grained and complex permission scenarios, that are difficult or impossible to implement with the minimal model. Unfortunately, the work behind ACLs never became a POSIX formal standard, and at the time of writing there's a wild mix of implementations with subtle differences and incompatibilities. We aren't going to explain how they work, we just say that they can be applied to files and directories, increasing flexibility and security.

For our purposes, the worst problems come when we have to preserve permissions in a distributed system: it's very difficult to implement a system able to preserve as much information as possible. There are a number of complications that make the operation prone to implementation errors, especially when we have different kernels and file systems. The semantics of ACLs differ widely among UNIX systems alone, not to speak of non-UNIX ones. A full ACL support over any kind of distributed system requires a mechanism so that all access decisions are performed in a way that honors ACLs: this means that every remote site should have a system-ACL support. Using only fs-ACLs will will lead to interoperability problems, although the good part is that they are automatically enforced on the end systems. In a Grid environment, there's the need to translate the resource-dependent ACLs in a common format.

In a Grid, we need to map a global security mechanism into a local one, which is independent from the "Grid security infrastructure". This brief discussion on ACL wants to remark the fact that every Grid resource should expose its security capabilities because not all of them are able to enforce the security and privacy requirement of some data types, due to the lack of security potentiality.

A PEP implementation can be used to map a Grid-ID in a local account, using File System ACL to enforce the Grid Authorization response. For our purposes, an example of an SRM implementation that can act as a PEP is StoRM [4].

4 Building a Grid Data Access Framework

Grid middleware should define a Grid security framework, encompassing both authentication and authorization in a standard way, and interfacing with local storage elements. Ensuring integrity, confidentiality and interoperability between heterogeneous systems can be achieved using a Web Service Architecture [22], which is an incarnation of a Service Oriented Architecture (SOA) in the context of the World Wide Web. SOA is the leading architectural style for building the current and future generation Grid technologies. Protocols based on Web Services provide important benefits for Grids, particularly in avoiding the tendency that proprietary binary protocols frequently become closely tied to particular implementations or languages. As stated in [10], "the Grid authorization model should be built on top of upcoming standards in the area of authorization, e.g. XACML, SAML, and WS-Authorization".

We think that a Grid authentication model should include an attribute authority that issues attribute assertions, and that a Grid authorization model should be built over a standard policy language. Different Policy Points should make decisions based on initiator identity and attributes, and so what is needed is a standard attribute language, that allow for interoperability between AA's and and PDPs. In addition, we have to remember that there may be several authorities that assert attributes for users, including other users. In section 3.2 we have already outlined the Policy Points actions, using the pull and push models. We should extend these variants thinking at the interactions between an authorization mechanism and a AA. From a technological point of view, a number of methods for requesting and encoding attributes already exist: for example X.509 Attribute Certificates[15], SAML[19] Attribute Assertions and XACML[20] Attributes. Since the emerging of the use of XACML for policy expression and the capabilities of SAML for attribute encoding, we should be able to combine this upcoming standards building a Grid Data Access Framework.

5 Conclusions

In the past sections we explored the technologies behind security in a Grid environment, focusing on the Grid Data Management Systems security aspects. With this paper, we didn't want to propose a definite solution, instead we defined some of the requirements and boundaries that we'll guide our future works in this field. In the following months, we will outline a Grid-based RBAC model for accessing distributed data, and we'll follow the implementation of a multipolicy authorization framework, based on XACML and SAML specifications. We'll define policies applicable to GDMS, their distribution and consuption, and interactions with monitoring and accounting services. At the same time we'll study methods to increase the performance of the whole authorization system.

References

1. J. Gu A. Shoshani, A. Sim. Storage resource manager: Essential components for the grid. 2003.
2. D. Chadwick. An x.509 role-base privilege management infrastructure. Technical report, 2002.
3. D. Chadwick. Authorization in grid computing. *Information Security Technical Report*, (10):33–40, 2005.
4. E. Corso, S. Cozzini, F. Donno, A. Ghiselli, L. Magnoni, M. Mazzucato, R. Murri, P.P. Ricci, H. Stockinger, A. Terpin, V. Vagnoni, and R. Zappi. Storm, an srm Implementation for lhc Analysis Farms, Computing in High Energy Physics. In *In Proceedings of the International Conference on Computing in High Energy and Nuclear Physics (CHEP2006), Mumbai, India*, Feb 2006.
5. S. Gavrila D.R. Kuhn R. Chandramouli D. Ferraiolo, R. Sandhu. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)*, (3):224–274, 2001.
6. E. Ferrari E. Bertino, P. A. Bonatti. Trbac: A temporal role-based access control model. *ACM Transactions on Information and System Security (TISSEC)*, (4): 191 – 233, 2001.
7. A. Caltroni et al. *G-Pbox: a Policy Framework for Grid Environments*. INFN Grid-it.
8. Alfieri et al. Voms, an authorization system for virtual organizations. In *In proceedings of 1st European Across Grid Conference*.
9. L. Pearlman et al. The community authorization service: Status and future. In *In proceedings at CHEP03, March 24-28 2003, La Jolla, California*.
10. Nagaratman et al. Security architecture for open grid services. memo GWD-I, GGF OGSA Security Workgroup, 2002m revised 2003.
11. Y. Demchenko et al. *Job-centric Security model for Open Collaborative Environment*, pages 69–77. IEEE Computer Society, 2005.
12. A. Grunbacher. Posix access control lists on linux. In *Submitted for publication at the USENIX ATC, San Antonio, Texas, June 2003*.
13. S. Tuecke I. Foster, C. Kesselman. The anatomy of the grid: Enabling scalable virtual organizations. *International J. Supercomputer Applications*, (15(3)), 2001.
14. A. Frohner P. Kunszt. glite data management security model disussion, 2005.
15. R. Housley S. Farrel. Rfc3281: An internet attribute certificate profile for authorization. Technical report, 2002.
16. EGEE security JRA3. Global security architecture. 2004.
17. The Globus security team. Gt 4.0 security. http://www.globus.org/toolkit/docs/4.0/security/, 2005.
18. M. Steenbakkers. Guide to lcmaps version 0.0.23. http://www.dutchGrid.nl/DataGrid/wp4/lcmaps/edg-lcmaps_gcc3_2_2-0.0.23/, 2003.
19. OASIS SAML TC. Oasis security assertion markup language (saml) tc. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security, 2005.
20. OASIS XACML TC. Oasis extensible access control markup language (xacml) tc. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml#XACML20, 2005.
21. OASIS XACML TC. Saml 2.0 profile of xacml v2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-saml-profile-spec-os.pdf, 2005.
22. W3C WG. Web services architecture. http://www.w3.org/TR/ws-arch/, 2004.