XPOLA – An Extensible Capability-based Authorization Infrastructure for Grids

Liang Fang and Dennis Gannon
Computer Science Department, Indiana University, Bloomington, IN 47405
Frank Siebenlist
Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL 60439
{lifang,gannon}@cs.indiana.edu, franks@mcs.anl.gov

ABSTRACT

There is great need for a secure, fine-grained, efficient, and user-friendly authorization infrastructure to protect the services in Grid community. Grid users and administrators still have to deal with authentication and authorization issues in the traditional supercomputer-centric fashion, especially with the host account maintenance and certificate management. This paper proposes a capability-based infrastructure that provides a fine-grained authorization solution to Web service deployments, and also manages to hide complex security issues from regular Grid users. Furthermore, it gives the resource providers and administrators the extensibility, flexibility and convenience to enforce their authorization policies at the resource with minimal efforts.

1. INTRODUCTION

1.1 The Grid and Its Security

Since the end of the last millennium, the Grid technologies have profoundly changed the way scientists compute by supporting collaboration and resource sharing securely across multiple domains. Interests from academia and industry in Grid technologies lead to a series of frameworks and applications, such as the Globus Toolkit and e-Science [6, 11]. The emerging Grid technologies are leveraging the Web services efforts, and are guided by Open Grid Services Architecture (OGSA) at the Global Grid Forum [29].

To realize the resource sharing across multiple domains, the underlying security infrastructure plays a key role. The widely used Grid Security Infrastructure (GSI), first integrated in Globus, provides secure communication, mutual authentication, and coarse-grained single sign-on (SSO) access to remote resources [7, 27]. It allows a user to access her resources across administrative domains through the use of her proxy certificates, and by delegating her rights to remote agents that manage remote resources on the user's behalf. The assumption it is based on is that the user had an account on the remote host and the remote agent is able to locate a mapping from the user's Grid identity to an associated local host account. This model has worked reasonably well for large, centrally coordinated Grids like NASA's Information Power Grid [12] and NSF's TeraGrid [3]. However, as the size and diversity of these research collaborations grow larger, it becomes increasingly difficult and sometimes impossible to stay with this supercomputer-centric model. For example, when a project has an educational outreach program that encourages thousands of grade school students to use services provided by the research Grid, we cannot assume these students all have Grid accounts on central resources. Indeed, for any research collaboration that does not have a strong centrally managed organization that can exert influence over resource providers, creating user accounts and managing Grid tools like Globus can be a serious administrative problem if it involves more than a few machines.

Specifically, in order to obtain access to a specific remote resource, even just temporarily, a Grid user-to-be has to go through the following steps:

First, she needs an X.509 certificate issued by a certificate authority (CA) that is trusted by the remote host. The user is supposed to manage her private key securely, and make those credentials accessible to the Grid-enabled client software.

As an alternative, a Grid portal solution provides the interface to Grid services through web servers and standard web browsers. This is definitely seen as a relief by the users because it frees them from the Grid security configuration pains. However, the user is still responsible for loading her certificate from a Grid-enabled host into a credential server, such as MyProxy, which stores the Grid users' credentials. From the user's credentials, the MyProxy server derives short-lived proxy certificates [28] after receiving requests from Grid portals or other Grid-enabled applications [18].

The configuration complexity is mirrored on the remote host. The user will have to contact the system administrator for an account on the remote machine that she wants to use as a computing resource. After being given an account with a user name, she has to ask the Grid administrator to add the mapping entry of her username and her certificate's X.500 distinguished name (DN) into a gridmap file. At runtime, GSI will authenticate the user and map her Grid identity to the local account according to the gridmap file. In some cases, the user may also be responsible for setting up a separate hosting environment under that account.

The administrators have the issue of maintaining an exploding number of user accounts, of which many will only

be used for a short time, or never be used at all. As a result, a significantly number of allocated resources is wasted in this way.

One of the identified problems is that, in its authorization process, GSI does not follow the principle of least authority (POLA), also known as the principle of least privilege. The gridmap file authorization model is coarse-grained, and in most cases allows the users and the user's delegates more rights than necessary for their jobs. As a consequence, the effect of compromises is more severe than needed [15].

Authorization frameworks, such as the Community Authorization Service (CAS) [18, 19, 26], VOMS [1], Akenti [25], PERMIS [2] and PRIMA [14] provide a set of solutions to some of these problems. Their ability to manage fine-grained access control can limit the risks. Nevertheless, the resource providers are still required to account for the identity and activities of each "user".

Peer-to-Peer (P2P) technologies, such as remote file-sharing systems, present us with another use-case for collaboration at the other end of the spectrum. In these "Grids", users provide resources and services to anonymous clients in exchange for similar access from other users. The only security guarantee the service providers have is the vague assurance that the service they are providing is limited access to network bandwidth and personal file space.

1.2 Goals and Accomplishments

Given that many scientists and educators have access to powerful desktop systems and small servers which they control and are either open to the Internet, or can be made visible via a proxy server, it is natural to consider the problem of running a user-level, peer-to-peer collaboration Grid that allows a group of people to share access to a collection of research resources. Many collaboration activities already operate in this manner, but they use ad-hoc security models.

This paper describes an extension to the Grid Security Infrastructure that exploits the convergence of Grid standards with Web services standards. It enables one to build peer-to-peer Grids with reasonable security and that are scalable and dynamic in structure. The goal of this work is not only to provide a fine-grained authorization solution to Web services in Grids, but also to hide the security issues from the regular Grid users as much as possible, and to give the resource providers and administrators the flexibility and convenience to enforce and modify their authorization policies for the resources at runtime with minimal efforts.

The systems we describe have the following characteristics:

1. Grid resources are made available to users through *capabilities*: signed tokens that allow specific users limited access to specific remote resources. These capability tokens can take the form of documents stating that user *X* has the rights to interact with service *A* and the document is signed by *Y*, the provider of service *A*.
2. The types of services that are provided to users in this infrastructure are Web and Grid services. For example, the interface to remotely execute a specific application, or an interface to push or pull data from a specific stream, or an interface to see a directory of objects or other services. It is *never* assumed that the user has an account on the computers upon which these remote services execute. Remote services execute under the identity of the service provider. Typically this service provider identity is the identity of the scientist or engineer responsible for running or maintaining the service on behalf of his collaborators.
3. The entire infrastructure is built on a P2P chain-of-trust model: The extend of the capability that I am willing to provide to an unknown remote user is limited by the level of trust I have in providing access to that user. I demand that any user accessing my resource will present a capability token signed by me and that the community authority has authenticated the user that is presenting the capability. If the service that I am providing requires the use of other services to do its job, I am responsible for assuring those service providers that the capability level that I provide to third parties is within the bounds of the capability provided to me by them.

With the problems and goals described, we will first introduce the basis of the capability model in the following section. Then we will discuss several existing authorization systems in Grids. In section 3, our capability-based authorization infrastructure, XPOLA, will be presented in both macroscopic and microscopic views. Next, the application of our capabilities framework is discussed as it applies to our Web services framework, Grid services framework, Grid portals, and application factory services. Lastly, we will list a set of challenges and the corresponding work to do in the second phase.

2. THE CAPABILITY MODEL AND RELATED WORK IN GRIDS

2.1 Access Control Matrix, ACL and Capability

The idea of capability originates from Dennis in 1965 [4]. A capability is an identifier that carries a set of specific access permission policies on the referred objects.

	Resource 1	Resource 2	Resource 3
Alice	**Yes**	**No**	**No**
Bob	**Yes**	**No**	**Yes**
Carol	**No**	**Yes**	**No**

Table 1 An Access Control Matrix

Illustrated in Table 1 is an access control matrix, which shows all the access rights of the users to a set of resources. If we describe the table in the column order, we call it an *access control list* (ACL). For instance, the ACL of "Resource 1" covers Alice and Bob, but not Carol. If we describe the matrix by row, then each row holds the capabilities of a user. For example, the user Bob has the capabilities to access "Resource 1" and "Resource 3", but not "Resource 2". A fine-grained authorization model may choose either way when describing its access control policy definitions.

Coupled with the principal of least privilege, the capability model has the advantage over ACL in that it solves the *Confused Deputy* problem, of which ACL is incapable, as proved by Hardy [10]. The "deputy" here is a program that is being executed on behalf of other programs or people with appropriate permissions. The Confused Deputy problem happens when the program is somehow fooled to apply its permissions to something that it shouldn't do. One of the reasons is the "ambient authority" issue, which means the program need not and cannot name the specific authority that justifies the operations that it takes to sense or modify the world around it.

An old example from the Confused Deputy problem is that a printing program that audits the users' printing activities by outputting the printing information to a log file, purposely, is fooled to overwrite some important system file such as /etc/passwd, which leads to a security hole that allows attackers break in. The problem lies in the fact that the printing service works under two different authorities. One is representing the user to print his files; the other is for the system or system administrator to record users' printing activities. Under each authority, it is able to do anything more than what the user or the system administrator means. ACL does not solve the problem because if it looks up in the ACL, the program that is representing the administrator does have the authority to write /etc/passwd. However, in capability model, the deputy would have been given a set of capability tokens specifying what is allows to do with the printing service and the logging file. Without being granted the capability to write /etc/passwd, it will simply be denied by the system.

The association of access control with objects can be extended by defining a capability as the property of a user or process to carry out a particular operation. This concept was first used in computer architectures such as the Cambridge CAP, the Hydra operating system for C.mmp, and the Intel iAPX 432 architecture (see [13] for an excellent survey). Capabilities have also been embedded in programming languages, such as E [16], and they seem to be especially important for distributed systems. Furthermore, capabilities have been applied to the design and implementation of operating systems like EROS [21].

The Web and Grid services are loose-coupled, highly distributed systems. The different clients and services may be part of different trust domains without any initial trust relationships. The process of trust establishment, unlike the ones in the operating system level security, requires minimal costs on communication as well as service load and maximal flexibility on authorization policy administration. The capability model externalizes the authorization process from the service, and thus reduces a large portion of its load.

For those reasons, many Grid deployments could benefit from the capability-based model. The fact that each Web service has a standard definition document with all its operations, parameters, identifier, and location, allows one to specify detailed authorization policies that could be protected by cryptographic means. HP's E-speak was the earliest capability-related effort on Web services [24]. Unfortunately, Lacking of commercial success, its development was halted in 2001. In the Grid community, authorization frameworks like CAS and VOMS, have also adopted a capability model explicitly or implicitly, as we will discuss in section 2.3.

2.2 ACL-based Authorization Infrastructure in Grids

A cross-domain fine-grained authorization model usually comprises of the nodes of clients, resources, and authorities. In a typical ACL-based model, the authority is commonly on the resource side and part of the same administrative domain as the resources. The resource provider often has no control over the authority. In order to configure the authorization policy, the resource provider collaborates with the administrators for the users. Usually this process is not automated, can be cumbersome, and may involve many steps.

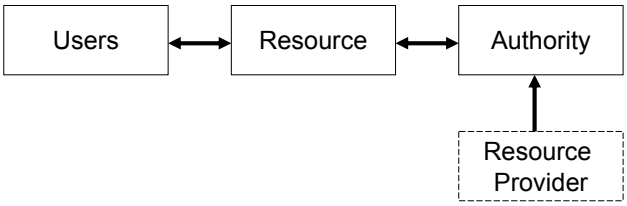

Figure 1 ACL-based Authorization Infrastructure

For now, we assume that the user has acquired her certificate and an account on the remote machine. After receiving a client's resource request, the resource node needs to verify the client's identity, after which it forwards the client's identity to the authorization authority for an access decision concerning the targeted resource. This decision will either permit or deny the client's request. PERMIS and Akenti have adopted the ACL-model that works in the flow illustrated in Figure 1 (note that Akenti can work in a capability-mode too).

Within this ACL-based model, the resource node is explicitly responsible for authenticating and authorizing

every single request. When the requests are repeated in a sequential manner from the same client, the authentication and authorization processes are thus repeated, which makes this system susceptible to scalability problems and possible Denial of Service (DoS) attacks. Akenti uses caching to mitigate the problem, but it hurts those isolated requests at the same time.

2.3 Capability-based Authorization Infrastructure in Grids

Another category of fine-grained authorization frameworks works with a capability model. In such infrastructures, the client's possession of a capability confers an access right to a resource, according to the pre-defined policy embedded in that capability.

The workflow is as follows: the user first sends a capability request to the authorization authority. The resource provider could either approve the request or delegate his rights to the authority to issue the requested capability tokens. Subsequently, the client sends her access requests along with the capability tokens to the resource node. On the resource node, the capability token's integrity is first verified. The policy details of the capability are then extracted as the reference to the final authorization decision. Figure 2 shows a simplified diagram of the capability model.

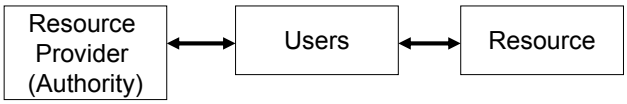

Figure 2 Capability Authorization Infrastructure

Besides the advantages on solving the Confused Deputy problem and externalizing authorization administration as we mentioned in section 2.1, the capability-based authorization infrastructure also has the benefit of reusability. The issued capability tokens are reusable for multiple accesses as long as their policy allows. It saves the repeated authorization evaluation processes that are inevitable in ACL-based systems.

A more practical reason is that some of the Grid users, especially those scientists, have the will to build their extemporaneous experiment services and allow others to use them securely without the lengthy interference of their system or Grid administrators. Capability allows them to distribute their services in a peer-to-peer fashion and remove the system security concerns that hinder their research.

CAS is an example of a capability-based authorization service. In CAS, a user must present a capability issued by user's community's authority to the resource provider to access the resource. The CAS capability token is bound to an extended proxy credential. Instead of the resource provider issuing the capability directly, the provider delegates part of his authorities to a community administrator. Every community has at least one central administrator to define the authorization policies and manage the service. The service authenticates the user with his X.509 certificate and issues capabilities to authorized users. On the resource server side, with the local policies, the resource provider still makes the final authorization decision, even if the user has been given the permission by the CAS server. However, the resource server will allow no more than what the capability allows.

Even though the CAS framework supports fine-grained authorization, resource providers have to verify that the CAS server is to be trusted and that the community's policies match their own local policies through some out-of-band form of configuration, which is not applicable in the case of highly dynamic services like the impromptu experimental ones that are provided by the scientist users.

Moreover, any form of centralized administration presents a single point of failure when being compromised or under attacks. One proposed solution of having multiple replicas of CAS servers to address the single-point-of-failure issue will statistically bring more chances of being compromised. Note that the capability model itself is not perfect either. Capability revocation is a thorny problem in CAS as well as any other capability-based systems.

Other fine-grained Grid authorization infrastructures are similar to CAS or Akenti. The different approaches can be distinguished in the way they bind policies to certificates and their authorities, while the enforcement mechanisms may differ slightly.

As we will show in the next section, XPOLA tries to keep the advantages of the capability model, while solving most of the issues associated with the capability model and other fine-grained authorization efforts. The main difference from any other existing capability-based infrastructures including CAS is its peer-to-peer administration model that securely brings maximal freedom to scientists and researchers. In the extreme case that all services are provided by one single user or a service account, that user becomes the administrator, just like the role in other infrastructures.

3. XPOLA, THE MACROSCOPIC VIEW

3.1 The Big Picture

The name of XPOLA stands for an eXtensible fine-grained authorization infrastructure that strictly conforms to the Principle of Least Authority (POLA). The design picture of XPOLA is shown in Figure 3. The capability manager (Capman) is the platform for resource providers to collaborate with their users. Through this platform, the resource users send requests to the providers for access rights; the provider processes the requests and creates the corresponding capability tokens. The capability manager stores the capability tokens and supports the providers with the functionalities for manipulating the capabilities: capability generation, destruction, update, renewal, request processing, and optionally pushing the capabilities to the users.

The registry service provides registration and discovery of the available Web service instances. It keeps the information of all the registered service instances. Every time a new service is instantiated, it is supposed to register itself by sending a Web Services Definition Language (WSDL) document to the registry. A WSDL document contains the detailed information needed to interact with the service instance.

If the capability of a specific resource for a specific user exists in the capability manager's storage, the user can fetch the capability from the manager directly and stores it in her local token agent. The token agent is an ssh-agent-like client for interacting with the capability manager and caching the retrieved capability tokens.

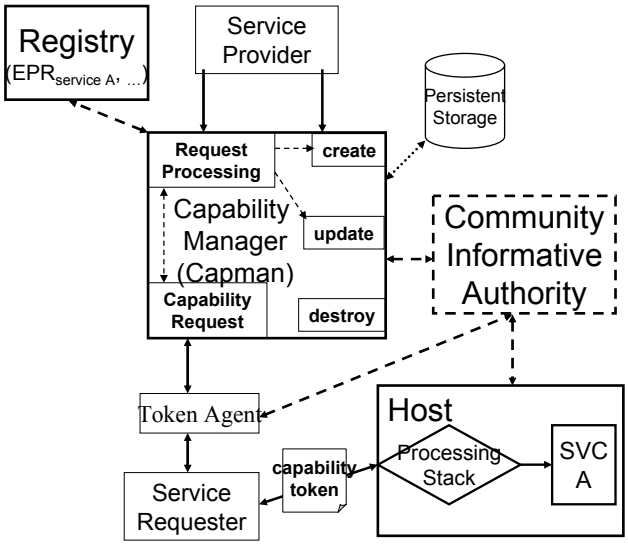

Figure 3 The Big Picture

The Community Informative Authority (CIA) is an optional trusted authentication service for identity verification in the community. It establishes trust relationships, either mutual or unilateral as required, between the users and providers. Note that CIA service does not make any authorization decisions for the providers. It is left to the providers themselves to make their authorization decisions based on the provided authentication information. CIA is not necessary if two parties can mutual authenticate each other by other means, for example, through the portal login authentication of a trusted portal. No matter whether it works implicitly or explicitly, the CIA service is designed to address the trust bootstrap problem in an XPOLA-enabled Grid community.

To make it better understood, let's walk through one of the use scenarios as a provider, *X*, of service *A*, and a service *A*'s user, *Y*. Suppose *X* first creates the service *A* on a remote host. Upon being created, the service A registers itself to a persistent registry. The provider then wants to distribute the service to other people including the user *Y*. Through a Capman service, he generates a set of capability tokens which contain detailed authorization policy and protected by *X*'s signature. We will dissect the capability token later, but here we just need to know that the identity of *Y* is included in the policy as one of the users. Now *X* advertises his service to *Y*, who will to use *A*. When *Y* tries to access *A*, his token agent contacts the Capman service for the required capability token. If available, the token is fetched and sent along with the service request to *A*, where the capability is to be verified. Thus the user *Y* is served by *A*, if the request matches what the capability allows.

3.2 Attack Analysis

Figure 3 can be simplified as Figure 4 to reflect the trust relationships between the main entities, namely the capability authority, the clients, the service instances, and an optional CIA service.

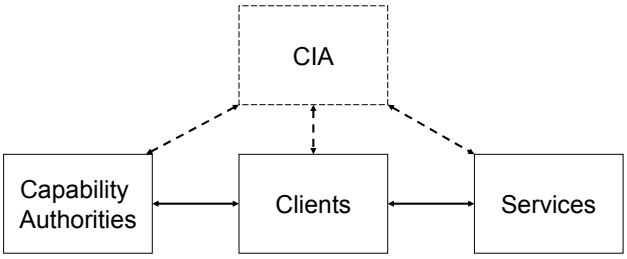

Figure 4 A Simplified Trust Relationship Picture for Attack Analysis

The service might be subject to various forms of illegal accesses and attacks from the clients. Examples of illegal or malicious access attempts are:

1. The user does not have a capability for the remote resource, or has a fake or invalid capability.
2. The capability is valid, but the user applies it beyond its policy definition. For instance, the user tries to access some other resource that is not specified in the capability policy.
3. The access is authorized, but the user orchestrates a Denial of Service attack (DoS) with the valid capabilities.

The first two situations can be handled easily by executing the capability enforcement. The last one is a tricky problem for capability-based systems. We need to revoke the malicious users' capabilities at runtime while the DoS attack takes place.

The capability manager does not need much protection as the capability tokens are inherently protected by the signatures from being forged or tampered. They are totally useless to any user who steals them, as the attackers' identities are not specified in the policies, unless a valid user's private key is available at the same time.

The worst case could happen is when the private key of a provider is stolen; however we can confine the consequences of this compromised authority to the provider

himself, who is merely one of the non-privilege users. Note that the providers won't be able to blame the administrators, because they themselves are responsible for the security of their own resources. XPOLA is flexible in that under some circumstances, administrators can take over the authorization work by running services with special non-privilege service accounts.

4. XPOLA, THE MICROSCOPIC VIEW

4.1 The Capability Tokens

A capability is a policy definition referring to a specific resource object. In XPOLA, a policy definition comprises of a delegation relationship and a set of authorization decisions for the referred object. The capability is protected with signatures and can be encrypted if needed.

Formally we define a capability as follows. Assume a resource provider named X creates a capability C to delegate a set of rights R on the service identified as S to a group of users P after the time of T with the duration Δt. The capability C_X is defined as

$$C = [X, P, R, S, T + \Delta t]$$

XPOLA has adopted the Security Assertion Markup Language (SAML) to express C, but it could be any policy languages, as long as the recipient, who himself set up the policy in most cases, is able to understand it. XPOLA is extensible to use policy definitions expressed in different formats or languages.

The policy definition for the operations of Web services is specified according to their Web Services Definition Language (WSDL) documents. The WSDL document is the standard interface description of a Web service.

The policy must contain the following items, corresponding to the formal definition:

- Parameter X: One or a set of the resource providers' X.500 distinguished names (DN) from their X.509 certificates.
- Parameter P: One or a group of specific users' DNs.
- Parameter R: A set of authorization decisions corresponding to the concerned operations of S.
- Parameter S: A service identifier, which is an endpoint reference (EPR) of a specific Web service.
- Parameter T: The lifetime of the capability.

Furthermore, the assertion will include signatures and verification methods. When privacy is a concern, the capability should be encrypted.

Within the Web services framework, when a user makes secure requests to remote resources, her client program communicates with them by exchanging SOAP messages that comply with Web services security specifications. Web services security is a series of emerging XML-based security standards from W3C and OASIS for SOAP-based Web services. The security related elements are added to the header section of SOAP messages, including signatures, references, confirmation methods, canonicalization methods, etc.

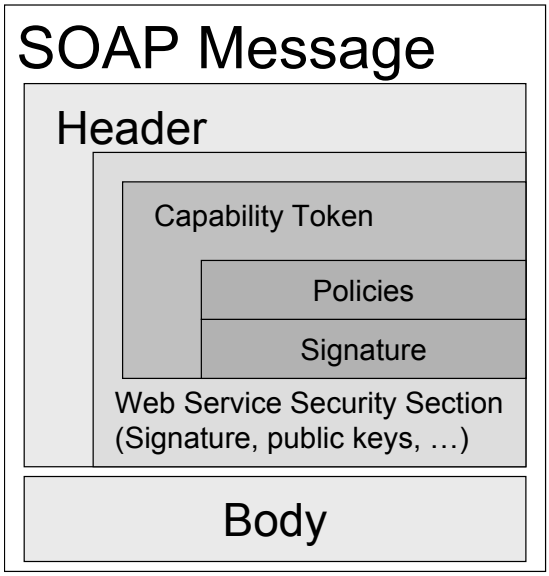

Figure 5 A Capability Bound to a SOAP Message

Similarly, capability tokens are embedded into the header section of users' SOAP messages, as showed in Figure 5. The SOAP message is then signed with the user's private key and the generated signature inserted into the Web service security section to protect the integrity of the SOAP message, along with the user's public key and identity, which will be verified against the capability token later.

4.2 The Capability Enforcement in Web Services

The enforcement is done when the capability-enabled SOAP message arrives at the remote resource service. We will identify the actual resource owner as A^*, the actual resource with the identifier S^*, the actual resource access attempt E that happens at the time T^*. The original capability C_A arrives at the resource as

$$C' = [X', P', R', S', T' + \Delta t']$$

The final authorization decision is made upon

$$C = C', X = X^*, S = S^*, E \subseteq R, T^* < T + \Delta t$$

and

$$\prod_{j \in C, C \subseteq R} r_j \sum_{r_i \in R} r_i \geq F_s. \quad r_i \in R, r_i \in \{0,1\}$$

Here, F_s is the authorization threshold of the resource S and $F_s = 1$. When $r_i = 0$, it means "denied"; while $r_i = 1$ means "granted". The set C is the crucial authorization decisions in R, which means any element r_j in the set C must be

"granted", in order that the access be authorized. The above process can be generalized as $r_i \in [0,1]$ and $F_s \geq 0$, when the provider is not so sure about his decision.

In XPOLA-enabled Web services, when a user makes a remote call from the client side, the client program first performs a preliminary authorization check by matching the capability policy with the SOAP body content and the user's identity. If nothing violates the policy, it inserts the appropriate tokens into the SOAP header, including the signatures; otherwise, the client is refrained from invoking the request.

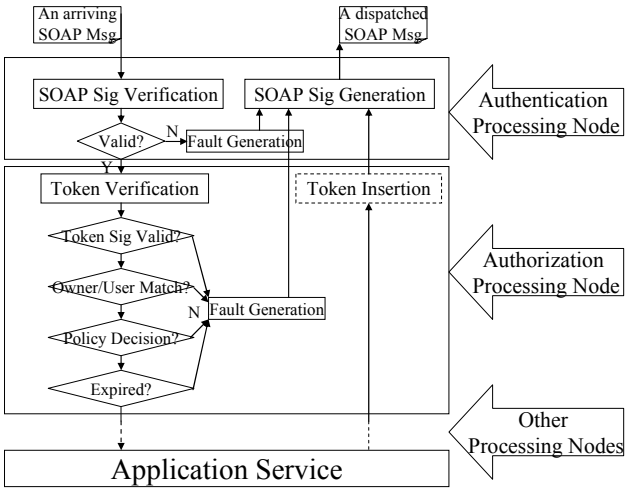

Figure 6 The Processing Stack on the Service Side

On the resource side, when the Web service receives a remote call through a SOAP message, the processing node first authenticates it by checking the validity of the user's signature against the whole message. If nothing is wrong, the authorization processing node verifies the signature of the embedded capability token. It then does a series of matching operations, such as the capability issuer's DN and the actual resource provider's DN, the actual caller's DN and the allowed users' DNs, the operations specified in the capability and the ones in the SOAP body. It also checks whether the capability has expired against the time. At last, it peels off the capability token and passes the rest of the SOAP message on to the next processing node. The detailed diagram is showed in Figure 6.

The client side policy checking prevents those unauthorized requests from reaching services and is able to prompt appropriate error information immediately to the users. Services thus need not waste their resources on processing these requests. The capability checking at the service side is mainly for guarding services from malicious users who elude the provided authentic clients to send the invalid requests directly. Such attack scenarios are discussed in section 3.2.

5. THE XPOLA IMPLEMENTATION AND APPLICATIONS

5.1 The Implementation

The implementation work includes the development of the core package of capability tokens with an application programming interface (API), the capability management toolkits, and their applications.

The capability's policy core adopts the popular XML-based security language, Security Assertion Markup Language (SAML) to express policy definitions [31]. SAML addresses three different use-cases.

1. **Single Sign-On.** The ability of a user (or subject in SAML terms) to authenticate in one security domain and have that authentication respected in another domain.
2. **Authorization.** The ability to ask questions about and describe the authorization of an actor to access a resources.
3. **Transactions.** The ability to pass the authority to complete a task from one security domain to another.

At its most basic level SAML is a language for making assertions about authentication, attributes, conditions and authorization decisions. For example, an authentication assertion typically takes the form "subject *S* was authenticated by means *M* at time *T*." Attributes are simply name-value pairs associated with facts about a subject. Authorization decisions take the form "It has been decided that subject *S* is authorized to perform action *A* on resource *R*. SAML provides a simple protocol to ask questions like "What authorization assertions are available for this subject?", "What are the values associated with these attributes for this subject?", "Is this subject allowed to access this resource in the following manner?".

In SAML terms, each capability policy document is a standard SAML assertion. The capability tokens are signed with the provider's private key. The public key is attached for verification.

XPOLA is designed with extensibility in mind. If needed, we can plug-in other XML-based policy languages such as eXtensible Access Control Markup Language (XACML) [32], or eXtensible rights Markup Language (XrML) [33]. The bottom line is, the provider can use any policy language he wants as long as he understands his own policies. Neither does XPOLA mandate the use of PKI to protect the capabilities. The provider may use symmetric keys to sign the capabilities, though in that case he needs some other ways like Kerberos, to authenticate the users and assertions.

XPOLA relies on XSUL to bind the capabilities to SOAP messages and to enforce the capability-based authorization. XSUL is a light-weighted SOAP engine and a general Web services framework [23]. Web service developers can write their own capability-enabled Web services with XSUL with limited effort. We have also integrated the XPOLA into GSX for building capability-enabled Grid services. GSX is an Open Grid Services Infrastructure (OGSI) [30] and Web

Services Resource Framework (WSRF) –compliant [34] Grid service framework based on XSUL [20]. The capability-based Grid service was assigned as homework in a distributed computing class. Turning an insecure Grid service into a capability-enabled one, was so simple that few student ever complaint.

The fact that our capabilities are inherently protected by themselves, allows us to use alternative ways to distribute them even through unprotected means. For example, the resource provider can use email, shared file system or even fax to deliver capabilities to the user.

To help the user, we provide a portlet-based capability manager and token agent build on the core capability APIs. We believe a user-friendly human-computer interface is vital in any security infrastructure. Many security systems fail, commercially or technically, simply because the users would not or could not follow the complicated rules. The capability manager as an independent Web service that shares the same database is also available.

5.2 XPOLA in Grid Portals

Grid portals are becoming very important in Grid deployments. They provide Grid community with a user friendly and familiar web-based interface to access their Grid resources. A Grid portal is made up of a cluster of customizable display units, called portlets. A dynamically generated web page in a Grid portal is the aggregation of a group of portlets in a personalized layout. The Grid portlets could act as the clients of the remote Grid services.

A Grid portal provides its own authentication mechanism through portal accounts bound with the users' proxy certificates. Both resource providers and users, as the portal users, share the same trusted portal context.

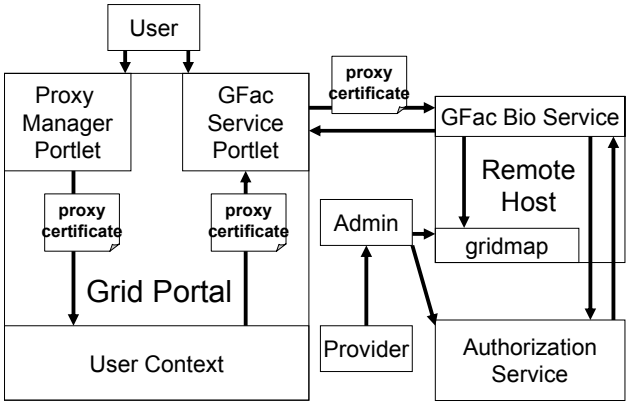

Figure 7 Authorization in an ACL-based Grid Portal

Traditionally, when a user logs into a Grid portal with her account, the portal requests a proxy certificate from a credential server, such as the MyProxy server [17], which returns a proxy certificate under the request. Usually the proxy certificate has a very short period of lifetime. The portal server stores the proxy certificate in the user context. While the proxy certificate is valid, the portal can fetch it when the proxy certificate's owner, the portal user, wants to invoke a remote Grid service through a Grid portlet. The proxy certificate goes along with the remote service call to authenticate the user, according the gridmap mechanism. The authorization steps, if available, are done in the ACL style, as shown in Figure 7.

One example of a Grid portal is the GFac portlet that works as the client of a GFac service. The GFac service is a Grid-enabled application factory service that encapsulates non-Web services, launches them on remote hosts at run time, and presents them as Web service instances [8]. GFac has been applied to launch complicated jobs remotely in the scientific domains of biology and meteorology.

Originally, GFac was a typical non-capability-based Grid service. After a GFac Bio service is launched remotely by its provider, a Bio service user can access the service from the Grid portal by contacting the remote service with her proxy certificate stored in her portal user context. On the remote host, her Grid identity is first mapped to a local account, where the authorization decision lookup is done by the GFac Bio service. With no choice, the GFac service provider has to delegate his authority to the administrator such that he can pre-configure the gridmap file and the authorization service, before allowing the designated users to access his service.

The XPOLA integration moves GFac's authorization process to the external Grid portal and avoids the administrator's authority brokering. After launching the service, the provider creates capabilities with capability manager portlet and stores them in portal user contexts. Later, when a portal user logs in, he can simply access the remote Bio service resource through the portlet client. The portlet automatically fetches the required capabilities from the user's context if they are available. The remote service authorizes the user according to the capabilities it receives.

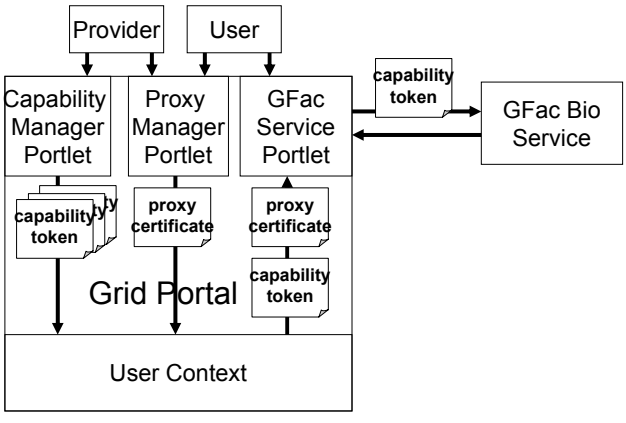

Figure 8 Authorization in an XPOLA-enabled Grid Portal

Because Grid portal is a trusted domain for all users, it implicitly plays the role of CIA by authenticating users with their portal accounts. Grid portal makes it possible to

transparentize the authorization process to portal users, as illustrated in Figure 8.

To summarize the scheme depicted in Figure 8: in a capability-enabled GFac service, the provider first launches a Bio job remotely through a GFac portlet. Then he simply creates a series of capability tokens for those authorized users with the capability manager portlet and stores them in the user contexts. After that, he can invite the users who have been endowed with the capabilities to use his service. The authorized user logs into the portal and she will be able to access the remote Bio service with the capabilities automatically fetched from her user context in the Grid portal; while as a naïve user, she may never know the existence of capabilities.

5.3 Performance

The capability-based authorization infrastructure is efficient as it allows for the reuse of capabilities and user-level administration. An authorization decision, once made, can be reused for multiple times.

However, if we only look at the narrowest definition of performance, it takes about 1000 to 2000 mille-seconds for a roundtrip communication between a client and a service, as shown in Figure 9. As the number of invocations grows, we see a slightly better throughput, probably due to the internal optimization of Java Virtual Machine itself, but it still falls in the same range. Considering the bulky SOAP header with security-related SAML assertions, signatures, and public keys, we are not too surprised about these observations. The reason seems to be a general problem of the implementations on Web services security nowadays: the underlying XML-related operations and processing, especially the canonicalization operations, are very expensive [22].

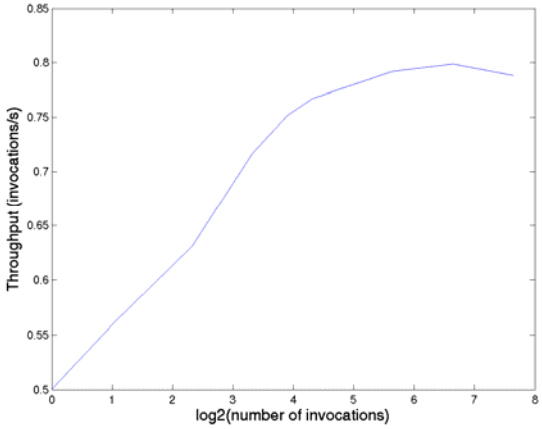

Figure 9 Throughputs of an XPOLA-enabled Web service

6. CHALLENGES AND FUTURE WORK

6.1 Session-based Communication

Performance is a big challenge in XML-based security implementations. Fortunately there are often possible workarounds to address the high overhead of the XML processing. The initial results of some informal tests, gives an indication that a session-based communication may be a promising solution. With session-based protocols integrated in our communication stack, we expect them to lower the XML security processing overhead significantly. In the second phase of our XPOLA work, we plan to implement a session-based secure communication specification, based on WS-Trust, WS-SecureConversation, and on the implementation work that was previously done [5].

6.2 Revocation

Revocation is a challenge for all capability-based systems. After the users acquire the capability, the provider does not have any effective approach to revoke it. In most capability-based systems, a capability's lifetime is so short that revocation is not practical, while the short lifetime may also help limit the amount of damage in the case of compromise.

One possible solution that we are investigating is to tie the capability to established sessions. This would allow us to revoke a specific capability immediately, by simply invalidating the underlying session.

6.3 Denial of Service Mitigation

Capability itself is a good mechanism for dealing with Denial of Service (DoS) attacks for providing fine-grained authorization. One of the principals of DoS attack prevention and mitigation is using the cost model – a client has to pay proportionally to the amount of the accesses to a service. A capability can be regarded as a resource "currency note" to be used to pay for the service it refers to. Capability tokens also make it possible to load balance the authentication and authorization work to other machines. A more sophisticated method will be to issue or request for dynamic capability tokens in the situation of an overwhelmed service, which would match the session-based capability system design.

7. CONCLUSIONS

In this paper, we introduced an efficient and user-friendly capability-based authorization infrastructure, XPOLA. It is based on user-level, peer-to-peer trust relationships, and used for building secure Web services and Grid services with the support of fine-grained authorization policy enforcement. We presented both microscopic and macroscopic views of XPOLA, and discussed its applications. Lastly, we brought up some challenges and tentative solutions that are to be addressed in future work.

8. ACKNOWLEDGMENTS

We would like to thank Gopi Kandaswamy, Aleksander Slominski, and Yogesh Simmhan for their discussions and collaboration on the integration work of the factory service, XSOAP and GSX. We also appreciate Markus Jakobsson for his invaluable suggestions to improve the paper.

9. REFERENCES

[1] R. Alfieri, R. Cecchini, V. Ciaschini, L. dell'Agnello, A. Frohner, A. Gianoli, K. Lorentey, and F. Spataro. VOMS, an Authorization System for Virtual Organizations. In European Across Grids Conference, February 2003.

[2] D. W. Chadwick and A. Otenko. The PERMIS X.509 role based privilege management infrastructure, *Future Generation Comp. Syst. 19(2), pp. 27-289, 2003.*

[3] C. Catlett. TeraGrid: A Primer. http://www.teragrid.org/about/TeraGrid-Primer-Sept-02.pdf.

[4] J. B. Dennis and E. C. Van Horn. Programming Semantics For Multiprogrammed Computations. MIT Technical Report MIT/LCS/TR-23, 1965.

[5] L. Fang, S. Meder, O. Chevassut, and F. Siebenlist. Secure Password-based Authenticated Key Exchange for Web services. *ACM Workshop on Secure Web Services, Oct.29, 2004, Fairfax, VA.*

[6] I. Foster, C. Kesselman. *Intl J. Supercomputer Applications, 11(2):115-128, 1997.*

[7] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke. A Security Architecture for Computational Grids. *Proc. 5th ACM Conference on Computer and Communications Security Conference, pp. 83-92, 1998*.

[8] D. Gannon, S. Krishnan, L. Fang, G. Kandaswamy, Y. Simmhan, and A. Slominski. On Building Parallel & Grid Applications: Component Technology and Distributed Services. *In CLADE 2004, Challenges of Large Applications in Distributed Environments. IEEE Computer Society Press, Jun 2004.*

[9] D. Gannon et al. Building Grid Portal Applications from a Web Service Component Architecture, 2004.

[10] N. Hardy. The Confused Deputy, http://www.cap-lore.com/CapTheory/ConfusedDeputy.html.

[11] T. Hey and A. E. Trefethen. The UK e-Science Core Programme and the Grid, *Future Generation Computing Systems, Vol 18 page 1017, 2002.*

[12] W. Johnson, D. Gannon, B. Nitzberg. Grid as Production Computing Environments: The Engineering Aspects of NASA's Information Power Grid, HPDC 1999.

[13] H. Levy, Capability-based Computer Systems, Digital Press, 1984, now available at http://www.cs.washington.edu/homes/levy/capabook/

[14] M. Lorch, D. B. Adams, D. Kafura, M. S. Koneni, A. Rathi, and S. Shah. The PRIMA System for Privilege Management, Authorization and Enforcement in Grid Environments. *Proceedings of the IEEE 4th International Workshop on Grid Computing, 2003.*

[15] K. Keahey, V. Welch. Fine-Grain Authorization for Resource Management in the Grid Environment. *Proceedings of Grid2002 Workshop, 2002.*

[16] M. Miller, M. Stiegler, Open Source Distributed Capabilities, http://www.erights.org/index.html.

[17] J. Novotny, S. Tuecke, V. Welch. An Online Credential Repository for the Grid: MyProxy. *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.*

[18] L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke. A Community Authorization Service for Group Collaboration. Proceedings of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks, 2002.

[19] L. Pearlman, C. Kesselman, V. Welch, I. Foster, S. Tuecke,The Community Authorization Service: Status and Future, *CHEP03, March 24-28, 2003, La Jolla, California*

[20] Y. L. Simmhan, Grid Service Extensions, http://www.extreme.indiana.edu/xgws/GSX/

[21] J.S.Shapiro, J.M.Smith, and D. J. Farber. EROS: A Fast Capability System. SOSP, 1999.

[22] S. Shirasuna, A. Slominski, L. Fang, and D. Gannon. Performance Comparison of Security Mechanisms for Grid Services, *the 5th IEEE/ACM International Workshop on Grid Computing,* Pittsburgh, Nov. 8, 2004.

[23] A. Slominski, M. Govindaraju, D. Gannon, and R. Bramley. Design of an XML-based interoperable RMI system: SoapRMI C++/Java 1.1. *In Proceedings of the 2001 International Conference on Parallel and Distributed Processing Techniques and Applications, Las Vegas, NV, Jun 2001* http://www.extreme.indiana.edu/xgws/xsoap/

[24] Spring, Buranarach, Schrubb, and Zatuchnaya. E-speak Revisited. *Forum on Design Languages Sep 3-7, 2001, Lyon, France.*

[25] M. Thompson, et al. Certificate-based Access Control for Widely Distributed Resources. In Proc. 8th Usenix Security Symposium. 1999.

[26] V. Welch, R. Ananthakrishnan, S. Meder, L. Pearlman, F. Siebenlist, Use of SAML in the Community

Authorization Service, http://www.globus.org/security/CAS/Papers/SAML%20Feedback-aug19.pdf

[27] V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, S. Tuecke. Security for Grid Services, *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press, June 2003.*

[28] V. Welch, I. Foster, C. Kesselman, O. Mulmo, L. Pearlman, S. Tuecke, J. Gawor, S. Meder, F. Siebenlist. X.509 Proxy Certificates for Dynamic Delegation. *3rd Annual PKI R&D Workshop, 2004.*

[29] Open Grid Services Architecture, OGSA working group at Global Grid Forum (GGF), https://forge.gridforum.org/projects/ogsa-wg

[30] Open Grid Services Infrastructure, OGSI working group at Global Grid Forum (GGF)

[31] OASIS, Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V1.1 http://www.oasis-open.org/committees/download.php/3406/oasis-sstc-saml-core-1.1.pdf

[32] OASIS, Core Specification: eXtensible Access Control Markup Language (XACML) V2.0 http://docs.oasis-open.org/xacml/access_control-xacml-2_0-core-spec-cd-02.pdf

[33] ContentGuard, eXtensible Rights Markup Language (XrML) 2.0 http://www.xrml.org

[34] OASIS, “Web Service Resource Framework”, March 2004.