

# Fine-grained Security Management in a Service-oriented Grid Architecture

Andreas Hoheisel<sup>1</sup>, Stephan Mueller<sup>2</sup>, and Bettina Schnor<sup>2</sup>

<sup>1</sup> Fraunhofer Institute for Computer Architecture and Software Technology, Germany

<sup>2</sup> Institute for Computer Science, University of Potsdam, Germany

## Abstract

One major reason for the lack of acceptance of Computational Grids in the industry are security concerns. Conventional Grid security architectures, such as GSI (Globus Toolkit), focus on the service provider's perspective and do not cover all concerns of the service user. For instance the management of user credentials is delegated to services which are not under full control of the user (e.g. MyProxy). Another drawback is that common Grid security systems do not cover the fine-grained authorization of services, taking into account the methods, their parameters, and the message flow in deciding whether a user or another service is authorized to access the service. This is particularly important for Grid workflow systems.

This paper introduces the Yagsi Security Infrastructure, which uses fine-grained and role-based security mechanisms in combination with restricted delegation of privileges, in order to overcome the drawbacks of current implementations. The design of the new security architecture provides a simple and convenient integration of legacy web services without the need of modification.

## 1 Introduction

By virtualizing resources, such as computers, data storage or services in general, the Grid technology enables scientific communities or industrial users to collaborate in a dynamic and powerful way. Different institutions are mapped to a Virtual Organisation (VO) combining the resources under the terms of a Service Level Agreement (SLA) and making them available to a much wider and heterogenous user base. This leads to new challenges for instance in the topic of accounting and billing because users no longer belong to one real world organization. An important requirement which must be fulfilled in such a dynamic environment is security. Accounting can only be performed if the identity of a user is assured. Likewise resource providers will only grant access if they can define the terms of usage or if the confidentiality of their data can be guaranteed.

### 1.1 Use Case

In order to illustrate security needs we choose a typical use case from the movie production industry. Figure 1 shows two different roles, simplified from a media VO: A freelancer who's job is to perform post production on images

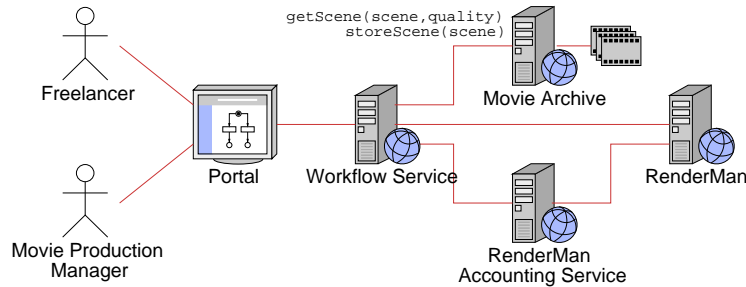


Fig. 1: Typical use case from the movie production industry that shows two roles, a portal and different web services.

with reduced resolution and a movie production manager keeping control of the whole process. Beside a portal and a workflow service there exists a movie archive service managing access to confidential movie data. Depending on the role it makes restrictions on the scene and the resolution of the images going to be checked out. Additionally there exists a RenderMan service which can be used directly by company internal users, whereas freelancers have to make their request through an accounting service.

## 1.2 Requirements

Upon this use case the following requirements related to access control can be identified: We need to provide Role Based Access Control (RBAC) which allows us to divide identities into groups like the *Freelancer* and to define access rules based on such a property. Furthermore, the trace of intermediate stations must be incorporated into the authorization decision so that the *RenderMan* service may verify that the service usage of external users is accounted. Finally the access control mechanisms must be fine-grained to the point of SOAP messages and their parameters. This allows the *Movie Archive* service to grant access on the base of the requested scene ID and the requested image resolution.

Additional requirements not covered by the use case are a restricted delegation of user credentials and a convenient integration. We want to be independent of the SOAP implementation and we want to provide security out of the box.

## 2 Related Work

For *authentication* the Public Key Infrastructure (PKI) represents a widely accepted standard. In contrast several, quite different approaches for *authorization* exist. This section briefly explains the security mechanisms used in two major Grid middleware platforms Globus[3] and UNICORE[2].

## 2.1 Globus

The Globus Toolkit 4 (GT4) combines security mechanisms within the Grid Security Infrastructure (GSI)[8]. GSI uses X.509 certificates to provide transport-level and message-level security on the basis of TLS respectively WS-Security[6].

Basic authorization in GSI is performed by a grid-map file. Specified in the deployment descriptor of a service this file maps the Distinguished Name (DN) of the user's certificate to a local user account. A more sophisticated approach is provided with the Community Authorization Services (CAS)[7]. It allows the resource providers to specify coarse-grained access control in terms of communities. Each community runs a CAS server which performs the fine-grained authorization. Also, in GT4 it is possible to assign the task of authorization to external services such as PERMIS[1].

In GT4 delegation is done on the basis of proxy certificates. In each delegation step a new certificate with a new key pair is generated. The DN of the original certificate is copied and slightly modified to indicate the proxy status. In contrast to ordinary certificates a proxy certificate is not signed by a CA but by the private key of the issuing certificate. Furthermore, the life time of a proxy certificate is usually restricted to few hours or even minutes. This proxy certificate can now be delegated to other services which have to act in the name of the subject the certificate refers to.

In order to provide a service with user credentials the GT4 Delegation Service [10] may be used. Located in the same container like the credential consuming service the user initially uploads his credentials to the Delegation Service. In response he receives an endpoint reference pointing to the credentials. Along with the request this reference is passed to the real service which may now fetch the credentials from the Delegation Service.

## 2.2 UNICORE

In comparison to the Globus Toolkit the UNICORE Grid infrastructure represents a top-down and job-oriented approach. Jobs are constructed by users on their Java-based UNICORE Client and are send to the Grid sites.

UNICORE introduces the Consigner/Endorser Model. It allows to distinguish between the identity which created a job (Endorser) and the identity which submitted a job (Consigner). Typically the user takes both roles but in the case of hierarchical jobs the Consigner may be the Network Job Supervisor (NJS) who sequentially submits the jobs to different sites.

In UNICORE the authentication is performed at the transport layer (SSL) on the base of the Consigner's signature. For the authorization additionally the Endorser certificate is used which is included in each job. The UNICORE User Database (UUDB) defines a mapping of these DNs to a local login.

### 2.3 Drawbacks of current implementations

A drawback of current security implementations is that they focus on the service provider's perspective and do not cover all concerns of the service user. For instance, the delegation solution provided by the GT4 Delegation Service is inconvenient and not flexible enough because credentials must be uploaded in advance and to each container hosting a credential consuming service. On the other hand a MyProxy[4] solution provides no fine-grained protection for the credential owner. Even if proxy certificates have a short life time, their range of application is not restricted. Without further acknowledgment any instance possessing such a certificate can act in the name of the user.

Another drawback is that common Grid security systems do not cover the fine-grained authorization of services, taking into account the methods, their parameters, and the message flow in deciding whether a user or another service is authorized to access the service. Also an ACL-based mapping of DNs to local machine accounts (grid-map file resp. UADB) is inconvenient and not flexible enough for a large scale VO.

## 3 The Yagsi Security Architecture

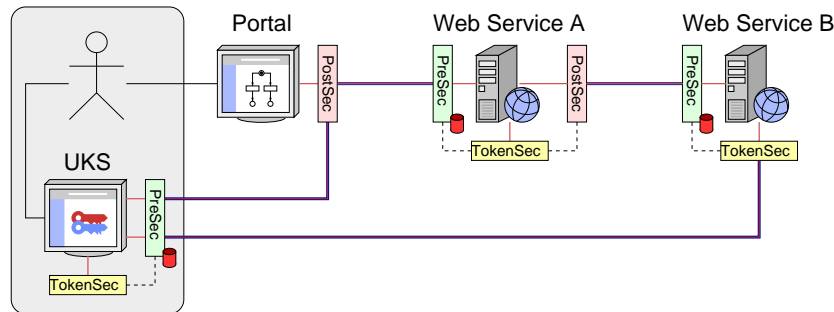


Fig. 2: Overview of the Yagsi Architecture which shows the User Keystore Service (UKS) and the security components PreSec, PostSec and TokenSec. Bold lines represent a secured SOAP communication. Dashed lines represent thread interaction.

Figure 2 gives an brief overview of the Yagsi security infrastructure. We use security components to protect web services and we provide a User Keystore Service (UKS) for managing user credentials. The WS-Security standard is applied to secure every communication between these components on the message layer. Special of the Yagsi approach is the use of security tokens which are attached to the SOAP messages and which are passed through all intermediate stations.

### 3.1 Security Token

A security token contains the user certificate and additional information such as an ID, a Uniform Resource Identifier of the User Keystore Service and attributes limiting the life time. Figure 3 illustrates a token that is initially signed by the private key of the user. When it travels through intermediate stations each station attaches its certificate and its signature.

The concept of a security token is essential for various reasons. From the service providers point of view it allows to perform an authorization decision based on the user querying a resource, even if the request was delegated by several intermediate stations. By collecting a trace of intermediates it enables the provider to restrict access based on the communication path. Finally it carries information from which entity user credentials can be obtained.

Considering the users perspective the token is an instrument to place restrictions regarding the lifetime of a request. Also it enables the user to verify whether a request for credentials was triggered by a former query of himself.

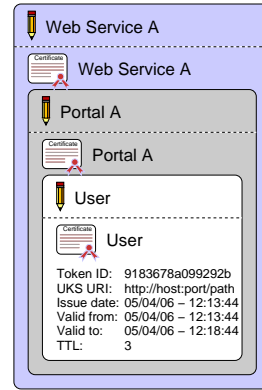


Fig. 3: SecToken

### 3.2 Security Components

The Yagsi security architecture distinguishes between three different security components: PreSec, TokenSec and PostSec.

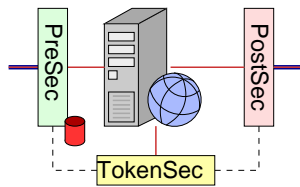


Fig. 4: Security Components

Each web service is protected by a *PreSec* instance which performs the authentication and authorization. On the basis of the WS-Security header the incoming request is validated and decrypted. Subsequently the SecToken is extracted and verified as well. The PreSec instance represents the Policy Enforcement Point (PEP) and the Policy Decision Point (PDP). It uses an XMLDB API to obtain policies needed for the authorization decision. Several PreSec instances may share one database. Likewise one instance may protect different services deployed in the same container.

If request is granted the message, reduced by the WS-Security header and the SecToken, is passed to the service.

Every Token which was received is stored in the *TokenSec* component. It signs the token with the private key of the station and it provides a SOAP interface for obtaining user credentials and information stored in the token. When credentials are needed the TokenSec instance contacts the UKS specified in the token. This communication again is secured by WS-Security mechanisms

likewise the token is passed with the request.

If a service needs to delegate a request the *PostSec* component is used. Implemented as a HTTP-Proxy this component attaches the SecToken and signs and encrypts the SOAP message.

All these three components run as different threads within one Java application and bind to different ports. Therefore this solution is independent of the SOAP implementation and the container the service is deployed into.

### 3.3 Restricted Delegation

Delegation of privileges is required when a service needs to perform a task on behalf of the user. For instance if input data must be read from or output data must be stored into a database the service needs user credentials, e.g. a password. Due to the distributed nature of the Grid environment, these credentials are usually delegated to the service in an automated way, so that no user interaction is needed.

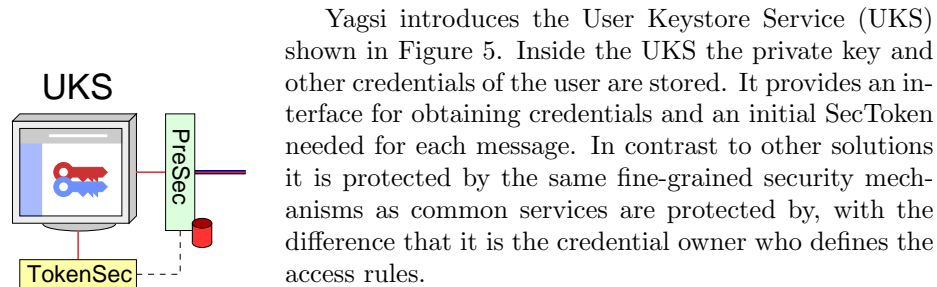


Fig. 5: User Keystore Service

Yagsi introduces the User Keystore Service (UKS) shown in Figure 5. Inside the UKS the private key and other credentials of the user are stored. It provides an interface for obtaining credentials and an initial SecToken needed for each message. In contrast to other solutions it is protected by the same fine-grained security mechanisms as common services are protected by, with the difference that it is the credential owner who defines the access rules.

The SecToken, which was created on the user side and which may have traveled through several intermediate stations, is passed with each credential request. This allows the user to grant access to his credentials in condition of a previous request of himself.

In order to allow multiple users to share one UKS instance all credentials are grouped by the DN of the user and can only be access if the DN delivered by the SecToken of the current request matches. Consequently each user has only access to a certain Policy within the PolicySet of the PreSec instance protecting the UKS.

## 4 Authorization

The PreSec component is responsible for performing an authorization decision. This section explains how Yagsi expresses and evaluates policies as well as how group membership is maintained.

### 4.1 Policy Language

Yagsi uses the eXtensible Access Control Markup Language (XACML)[5] for describing policies. It is well established and defines a Request-/Response- and

a Policy Language. The latter is divided into **PolicySets**, **Policies** and **Rules**. There exists a reference implementation from Sun Microsystems, SunXACML[14], which provides an API for the construction of requests and policies. It also ships with a Policy Decision Point which performs an authorization decision based on a Policy and a request.

In XACML every **PolicySet**, **Policy** and **Rule** owns a **Target** element which must match before any of these items can be applied. We map the **Subject** entry of the **Target** to the DN of the user, the **Resource** element to the service which is going to be invoked and the **Action** entry to the SOAP method called. To every **Rule** additional **Conditions** can be attached. In our case this is the group membership of the user, a list of mandatory intermediate stations and restrictions to the domain of the parameter list of a method.

## 4.2 Group Membership

The group membership mechanism was designed to match the requirements of dynamic and large scale VOs. Figure 6 illustrates the hierarchical approach with no restrictions to the depth of the tree. A group is identified by its name and the name of all parent groups prepended and separated by a slash (e.g. **Magrathea Movie/Production Manager**).

Users are assigned to groups by attaching their DN to all groups they should belong to. If a user is member of a subgroup he is also member of all parent groups. To even increase the flexibility we introduce group links which allow to form new groups by referring to existing ones. In Figure 6 this is done within the **Admin** group of **Project42**. This for instances leads to a membership of the user **Zaphod** to the groups **Magrathea Movie**, **Magrathea Movie/Admin**, **Project42** and **Project42/Admin**.

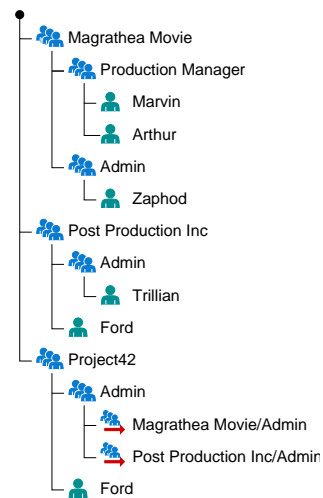


Fig. 6: Group Membership

## 5 Conclusions

In comparison to conventional security implementations Yagsi focuses on the security needs of the Grid resource provider and the Grid user. Stored in the UKS and secured by a PreSec component, credentials stay under full control of their owner.

Instead of using proxy certificates we attach a security token to the SOAP message in order to identify the user of a request. By signing this token in each delegation step it is possible to let the intermediate stations incorporate into the authorization decision. Further fine-grained access criteria are the SOAP

method including their parameter list and the group membership of the user. All policies are expressed and enforced using the OASIS standard XACML.

In order to secure web services, Yagsi introduces the three components PreSec, TokenSec and PostSec which may run outside the web container. This provides an solution independent of the SOAP implementation and simplifies the integration in existing Grid environments. After finishing a prototype in march 2007 we plan to apply Yagsi to national and international Grid projects such as CoreGRID[9], the K-Wf Grid[12], the MediGRID[13], and the Instant-Grid[11].

## References

1. D.W. Chadwick and A. Otenko: The PERMIS X.509 Role Based Privilege Management Infrastructure, *Future Generation Computer Systems*, 19(2):277-289, February 2003.
2. Dietmar Erwin: UNICORE Plus Final Report - Uniform Interface to Computing Resources Joint Project Report for the BMBF Project UNICORE Plus Grant Number: 01 IR 001 A-D, Duration: January 2000 to December 2002 ISBN 3-00-011592-7, 2003.
3. I. Foster: Globus Toolkit Version 4: Software for Service-Oriented Systems, *IFIP International Conference on Network and Parallel Computing*, Springer-Verlag LNCS 3779, pp 2-13, 2006.
4. D. Kouril and J. Basney: A Credential Renewal Service for Long-Running Jobs. *6th IEEE/ACM International Workshop on Grid Computing (Grid 2005)*, Seattle, WA, November 13-14, 2005.
5. T. Moses: eXtensible Access Control Markup Language (XACML) Version 2.0, *Tech. Rep.*, <http://docs.oasis-open.org/xacml/2.0/>, February 2005.
6. A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker: Web Services Security: SOAP Message Security 1.1, *Tech. Rep.*, <http://docs.oasis-open.org/wss/v1.1/>, February 2006.
7. L. Pearlman, V. Welch, I. Foster, C. Kesselman, and S. Tuecke: A Community Authorization Service for Group Collaboration, *Proc. of the IEEE 3rd International Workshop on Policies for Distributed Systems and Networks*, Monterey, CA, June 05-07, 2002, pp. 50-59, 2002.
8. V. Welch, F. Siebenlist, I. Foster, J. Bresnahan, K. Czajkowski, J. Gawor, C. Kesselman, S. Meder, L. Pearlman, and S. Tuedke: Security for Grid Services, *Twelfth International Symposium on High Performance Distributed Computing (HPDC-12)*, June 2003.
9. CoreGRID, <http://www.coregrid.net/>, Accessed: September 2006
10. Globus Alliance: GT4 Delegation Service, <http://www.globus.org/toolkit/docs/4.0/security/delegation>, Accessed: November 2006.
11. InstantGrid: Ein Grid-Demonstrations-Toolkit, <http://instant-grid.de>, Accessed: October 2006.
12. K-Wf Grid: The Knowledge-based Workflow System for Grid Applications, <http://www.kwfgrid.net>, Accessed: November 2006.
13. MediGrid: GRID-Computing fuer die Medizin und Lebenswissenschaften <http://www.medigrid.de>, Accessed: October 2006.
14. SunXACML: XACML Implementation of Sun Microsystems, <http://sunxacml.sourceforge.net/>, Accessed: October 2006.