Using Virtual Organizations Membership System with EDG's Grid Security and Database Access

Marko Niinimaki[1], John White[1], Wim Som de Cerff[2], Joni Hahkala[1],
Tapio Niemi[1], Mikko Pitkanen[3]
1 Helsinki Institute of Physics at CERN, CH-1211 Geneva
2 Royal Netherlands Meteorological Institute KNMI
3 Helsinki University of Technology

March 17, 2004

Keywords: Grid security, virtual organizations, Grid use case.

This paper describes the European Data Grid's (EDG's) java security system and Spitfire database access system giving special emphasis on the virtual organization technologies. These technologies create a feasible framework for authentication and authorization in distributed Grid applications.

A virtual organization (VO) is a collection of people in the same administrative domain. A user can belong to many virtual organizations and have a different role (user, client, administrator,..) in each of them. An authorization of a user to different services within a VO is based on the user's identity and a service called a Virtual Organization Membership Service (VOMS) that maps these identities with roles.

The user proves his identity over the internet using authentication process. The user normally authenticates using his credentials, which comprise of a certificate chain and a private key. In Grid systems, the user usually authenticates using proxy credentials that are derived from the actual credentials. The proxy credentials comprise of the user's certificate chain added with a proxy certificate and a proxy private key. In the proxy creation process, the user's VO information, including groups and roles, is included into the proxy certificate.

In order to use these proxy certificates with VO information we have created an authorization system and to demonstrate the usage we have extended the functionality Spitfire, a relational database front end. This involves assigning the user a database role (read, write, update..) based on the VO information in his certificate. There is also a GUI for configuring the authorization service. The earth observation team's database access for ozone profile validation is used here as an example of an application.

1 Introduction

Authentication (i.e. user identity verification) and authorization, i.e. deciding whether the user can have an access to a certain resource are essential in distributed applications. The Grid computing field can be characterized as a collection of heterogeneous computing resources that are shared by many individuals and organizations. This has given rise to the concept of "virtual organizations". A virtual organization (VO) is a collection of people in some administrative domain. A user's relationship with his VO is defined by the organization's internal hierarchy. The user can be a part of any number of internal groups in their organization and have multiple roles in many organizations [ACC+03]. A user is authorized to perform tasks on a computing Grid according to their VO affiliation and their role(s) within the VO.

The authorization process becomes more complex when the number of users and the number of possible roles of the users in the service increases. As services are usually distributed, a centralized service managing the authorization data is needed. This service is called a Virtual Organization Membership Service (VOMS) [ACC+03]. The VOMS service is a front-end to a database where the information about the users is kept. The server maintains lists of groups, roles and capabilities that belong to each user in that VO. The VOMS is used to bind authorization information to a users identity [ACC+03]. European DataGrid's (EDG's) VOMS is a mature application that utilises EDG's grid security functionality, based on certificates.

A Grid user, operating with Globus [FK97] middleware, uses the command `grid-proxy-init` to create a proxy certificate, which is made using their credentials. This proxy certificate can then be passed to other Grid resources as the user's credential. When EDG's VOMS service is used, the user invokes `edg-voms-proxy-init` program to contact the VOMS server. This program produces a proxy certificate, similar to the previous case, but also containing the authorization information from VOMS service [ACC+03]. When the user enters a service presenting his VOMS certificate, the service can find out the user's authorization information from the certificate. This removes the need to maintain up-to-date per-user authorization information in every server.

The information about a user's VO and their role in it can be easily utilized in the context of distributed databases being accessed by an EDG database application, Spitfire [Pro01]. In a typical case, if a Spitfire administrator knows a user's VO (e.g., a collaboration called NETG) and their role within this collaboration (say, an administrator), they can base the access decision on that VO, group or role. Using Spitfire, along with the edg-java-security package and a graphical user interface (shown in Figure 2), we can easily add a rule concerning the access rights of the VO, group or role. This greatly simplifies the task of the database administrators, since they do not need to describe access rights for each individual user, they can just describe it for user groups of roles.

Related work Foster, Kesselmann and Tuecke emphasize the role of virtual organizations in resource sharing in [FKT01], and mention database access in one of their Grid scenarios. The OgsaDai consortium [Ogs02] has defined an architecture for Grid

database accesses and released an implementation of it. However, as far as we know, the European DataGrid project has been the first to design and implement a VO-based database access management system.

Contents The rest of this paper is organized as follows: in Section 2, we present the technology used and the certificate-based security model that our VO implementation utilises. We describe, too, how Spitfire's database access works and discuss database access authorization in general. In Section 3 we discuss how certificates with VO extensions are used in combination with Spitfire using earth observation as an example. Finally Section 4 provides conclusions and some items for future research.

2 Technology and terminology

2.1 Spitfire and EDG java authentication

Spitfire [Pro01] is a project of Work Package 2 within the European Data Grid Project. It offers a Java servlet that accepts database requests, forwards them to actual relational databases and displays the results in XML. The EDG's authentication and authorization software is edg-java-security, and it analyzes a user's rights to execute queries based on the user certificate presented to the system. This authentication and authorization software is used with other EDG software, for example Spitfire.

In edg-java-security, the authentication is based on a hand shaking protocol in Secure Sockets Layer and Transport Layer Security. The server and client send each other their X509-format certificates and messages encrypted by their private keys, called challenges. These challenges can be verified by public keys included in the X509 certificates. This way the server and the client authenticate themselves as owners of their respective certificates [Sec03]. The authentication sequence is as follows. First, the user gets the certificate of the server. The client program then checks that the server's certificate is signed by a trusted certificate authority (CA). This is done by decrypting the certificate's signature with the CA's public key. Moreover, the client verifies and the server responds to the its challenge by encrypting it with its private key. After this, the user knows that the server is who it claims to be and the client encrypts the random data the server sent with his private key and sends his certificate chain to the server along with the encrypted random data, see [FK97]. The server then checks the user's certificate chain and the encrypted data. If the server is able to decrypt the challenge, with the user certificate's public key, it can be sure that the user is the same as the owner of the certificate or proxy certificate.

The proxy certificate enables a single sign-on process by "representing" the user to Grid services. When a proxy certificate is used as a credential, the user sends their certificate and the proxy certificate to the server [FK97]. The proxy is signed by the user and the user's certificate is used to verify the proxy's signature. This way the chain of trust is delegated to the proxy. The proxy certificate can be used by the user for access to various services, as it carries the user's signature as identification. However, the only apparent feature of the proxy is its issuer, i.e. the user's certificate subject like "O=Grid, O=NorduGrid, OU=hip.fi, CN=Joe User". Each Grid service needs to

decide independently the access rights of each certificate owner. This can be improved by introducing extensions to the certificate; in our case a VO extension that states the user's VOs, groups and their role in each of them.

2.2 Authorization and Relational Databases

With EDG java authorization mechanism four levels of authorization granularity can be implemented. First two do not require any support from the actual web service the authorization mechanism protects, the last two require support from the web service.

Fist, the most coarse grained authorization just decides if the user is allowed to access the service. This is simple to configure and doesn't require any knowledge of the service.

Second, the method based authorization decides if the user is allowed to access a certain method in the service. In this system local roles are defined for the service and methods are grouped for these roles. For example "administrator" has access to all methods but "read" has access to the read and list methods. The configuration is a little more complex than the first case.

Third, the service role based authorization relies on the service that enforces role based authorization. For example in database access the databases usually have role based access control with a table level granularity. In this method the authorization groups and roles are mapped into the database roles and thus the database administrator can define access policies for individual tables. This case requires full support for the authorization mechanism in the service, but configuration is of the same complexity as in the previous case.

Finally, the fine grained authorization requires the service to enforce fine grained authorization. In this case every object has a reference to an access control list (ACL). For example every row in database has an ACL and every file in storage element (SE) has an ACL. In this case the client software needs to support the ACLs and the users should know about and control their ACLs for them to be useful. This is the most complex case as it requires support from every level of the system and in only the most simple and limited cases can the configuration be automated.

Authorization and security are essential in client-server database systems. In this section we discuss briefly how they are implemented in SQL databases. We follow the book by Elmasri and Navathe [EN94].

In general, two methods are used in database access control: discretionary and mandatory access control. In discretionary access control different privileges for database objects (e.g. tables = relations, columns = attributes) are granted to the users while in mandatory access control the data and the users are classified in different security classes. A user, in order to view the data, must have the same or higher security class than the data in question. In the following discussion, we will refer to discretionary access control methods since almost all relational database systems use it while the mandatory control method is used only in some special systems. Moreover, the SQL standards support only discretionary access control.

In SQL, privileges can be assigned to the account (user) level or the database object (relation) level. At the account level the privileges define what operations a particular user can perform in general, and in the database object level the privileges specify the

operations a user can perform on the object (e.g. select, modify). In order to perform an operation, the user must have both account level and the object level privileges.

The basic privileges for relations (tables) are *select*, *modify*, and *reference*. The select privilege allows the user to retrieve data from the relation and is defined only on the relation level; views can be used to allow only some attributes to be retrieved. The modify privilege allows the user to modify the data and can be defined in a more detailed manner as *update*, *delete*, and *insert* privileges. The modify privilege is also defined on the relation level, and the update and insert privileges can also be given on the attribute (column) level. The reference privilege allows the user to define references to a relation, e.g foreign key constraints.

SQL has *grant* and *revoke* commands for defining privileges. With the *grant option*, a privilege can be given to the user so that he can grant it further. SQL also supports roles. The role is "a set of privileges" that can be assigned to the user. This makes administration easier since several privileges do not need to be granted separately to the user.

2.3 VOMS

Essentially the Virtual Organization Membership System (VOMS) presents an extension to a user's X509 proxy certificate, that includes their VO membership information. When a VOMS-proxy is generated with the `voms-proxy-init` command is used, the VOMS server is contacted to request a VOMS block which will be included into the user's proxy certificate. That proxy certificate follows the standard X509v3 [HFPS99] certificate format. All the standard fields of the proxy certificate are used to store the user's authentication information. An additional extension (1.3.6.1.4.1.8005.100.100.1) is used to include VOMS block in the user's proxy certificate. The authorization information is stored in triplets with the following syntax:

```
GROUP: string
ROLE:  string
CAP:   string
```

When users wish to use a grid service, they pass their credential (in this case the VOMS-extended proxy certificate) to the service interface, for example, Spitfire. The grid service then extracts the user's authorization information from the extension part. The extensions part of the proxy certificate also includes the VOMS signature and validity period of the role mappings. The VOMS signature is used to to verify that a trusted VOMS service has attached the authorization information to the user's proxy certificate.

The actual information that the VOMS extension contains can be configured by the VO's administration. An example that demonstrates VO admin user interface is shown in Figure 1.

In the future, it is foreseen that the VOMS block will become a full Attribute Certificate [FH02] which stores the authorization information. This will not change the way information is used in a service since the same information is extracted from both formats.

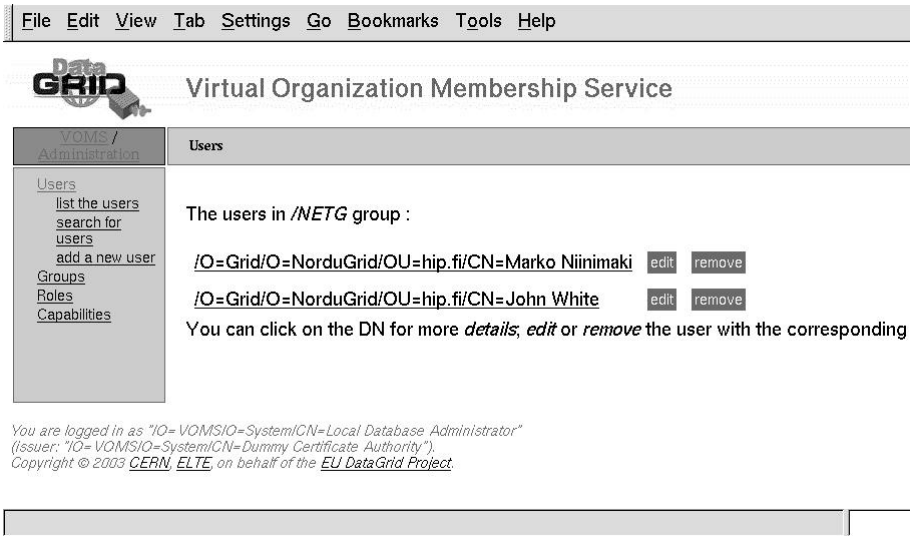

Figure 1: VOMS web interface

3 Combining VOMS, security and database user authorization mechanisms

In the Spitfire service, database roles are divided into categories of “read”, “write”, “update”, and “create” by default. A user that has been assigned the read role is allowed to browse the database in question; similarly, “write” role owner can insert data, “update” role owner can modify data, and “create” role owner can create new tables. Normally, of course, a user can have many roles; for instance in the case of a database administrator all of the above.

A *policy* is a collection of mappings of roles and users. Policies are designed by EDG’s TrustManager software components. For instance, policy “test” (see below) is based on a regular expression mapping that allows roles “read”, “write”, “update”, and “create” to a person whose certificate subject is “O=Grid, O=NorduGrid, OU=hip.fi, CN=Joe User”, and “read” to persons whose certificate subject contains “O=Grid, O=NorduGrid, OU=hip.fi”.

Spitfire can access any relational database with a Java connector, but the default implementation is based on MySQL [MyS01]. With MySQL, the roles are applied on database-wide basis.[1] Different VO’s can have their databases accessed through a single Spitfire service and the access configuration is managed with standard Java database access methods.

Figure 2 shows a web browser-based graphical user interface that is used to administer a Spitfire installation. This tool is used to modify the XML file that stores the user-to-role mappings within security policies. The tool contains an editor (see

[1] MySQL’s “database” roughly corresponds to Oracle’s tablespace. Many database vendors, including MySQL and Oracle, implement at least “per database” and “per table” access control.

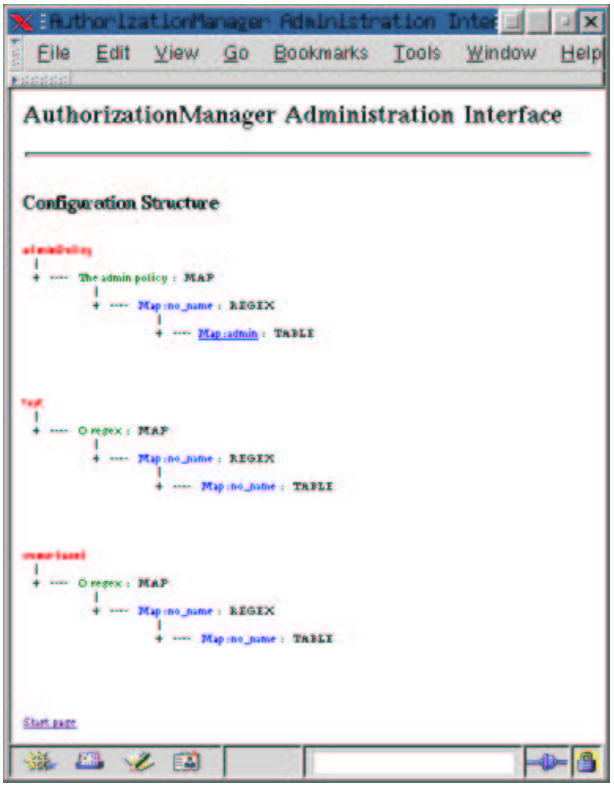

Figure 2: AdminGUI interface

Figure 2), which can be used to view and modify the XML-based role and policy definitions. In order to use this tool, an administrator of a Spitfire service must have a valid certificate loaded in their web browser and this certificate must map to an "administrator" role within the "adminPolicy". The third policy in Figure 2 shows policy "voms-based" that will be discussed in the use case.

3.1 Use case: Ozone profile validation

Validation of satellite data using ground-based measurements is common practice within the Earth Observation community. One of the use cases of Earth Observation work package (WP9) within the EDG project is the calculation and validation of ozone profiles. The ozone profiles are calculated from satellite measurement data. Ozone Profiles contain measurements of ozone within a vertical column of atmosphere at a given latitude / longitude location above the Earth's surface. The profiles are validated using ground based light detection and ranging (LIDAR) measurements, which can also measure ozone profiles. The validation consists of comparing the ground-based measurement profiles with profiles derived from the satellite data, coincidence in time and

location.

As the satellite data has global coverage over a time span of 5 years and ground-based LIDAR measurements are point measurement campaigns, databases are used to store metadata for both LIDAR measurements and calculated ozone profiles. Especially the date-time and location (lat/lon) data is important, but also the Logical File Name (LFN) and quality parameters are stored. In this way coincidence data can be found by querying the database. Note that the data itself is stored on GRID storage element (SE)s and can be found using the LFN stored in the metadata database. For accessing the metadata database Spitfire is used.

From the use case three distinct user-groups with different access rights can be derived: the producers of the ozone profiles who need to be able to insert their profile metadata, LIDAR data providers who need to be able to insert their LIDAR metadata and validation experts who need read access to the metadata. We also need a database administrator to set up the database. Spitfire with VOMS capability can provide all the required functionality.

Spitfire offers standalone Java and c++ clients for accessing the service. First the user requests a VOMS certificate with `edg-voms-proxy-init` command. The certificate is written and stored to a file ”/tmp/x509up uid” (where id is the user’s identity number in the computer). When the client accesses a service, the VOMS (proxy) certificate is sent to the service by Spitfire client. The service is then able to extract the authorization information from the certificate.

After the user sends the query to the service, the service finds out which VO the user belongs to. Further, the role of the user in his VO is also extracted from the certificate. With this information the service is able to define the user’s access rights. This removes the need from a service to maintain specific user ids and their mapping to database access rights.

In the case of successful authorization, the result of the user’s query is returned to the client. An extract of sample client code is shown in Figure 3, where the user selects and prints all LFNs of profiles coincidence with his LIDAR data measured at the ground station located at Haut Provence (see the selection criteria, Haut Provence: HP):

The security and authentication features are completely invisible to the programmer. An example of running the code and displaying the results shown in Figure 4. There, it can be noticed that the user requests policy ”voms-based” and role ”read-test” and uses his proxy certificate named ”/tmp/x509up_u513”.

4 Conclusions

In this paper, we have presented a design and an implementation of virtual organizations and database access. The main benefits of the design are: (i) it separates authentication and role authorization in manageable modules; (ii) There is no need to enter user id - password -pairs because of certificates; (iii) The database access is generic, i.e. can access relational databases of different manufacturers; (iv) It outputs XML that can be easily processed further.

```
public final class SelectLFN {

    public static void main (String [] args) throws Exception {

    URL base = new URL("https://datagrid.nadc.nl:8443/Spitfire/services/SpitfireBase");
    SpitfireBase sfBase = new SpitfireBaseServiceLocator().getSpitfireBase(base);

    try {
        String HP =" Lat < 48 and Lat > 38 and
             Lon >  0 and Lon < 10 and
     DateTimeStart > \"1999-06-01 00:00:00\" and
     DateTimeStop < \"1999-06-31 00:00:00\" ";

        SpitfireResult result = sfBase.simpleSelect("gome",
                                                    "GOME_OPERA",
                                                    "LFNOutput",
                                                    "HP",
                                                    0);
        SpitfireResultHelper.printResult(System.out, result);


    } catch (Exception x) {
      System.out.println("SelectFromLFN ERROR: " + x.getMessage() );
    }
  }
}
```

Figure 3: Sample code: SelectLFN.java

Moreover, we have demonstrated its usability by a concrete example used in Earth Observation group's ozone validation application.

References

[ACC+03] R. Alfieri, R. Cecchini, V. Ciashini, L. dell'Agnello, A. Frohner, K. Lorentey, and F. Spataro. VOMS an authorization system for virtual organizations. In *Proceedings of the 1st European Across Grids Conference - Santiago de Compostela, Spain, 13-14 February 2003*, 2003.

[EN94] R. Elmasri and S. B. Navathe. *Fundamentals of database systems (2nd ed)*. Benjamin / Cummings, 1994.

[FH02] S. Farrell and R. Housley. Rfc 3281, an internet attribute certificate profile for authorization. Available on http://www.ietf.org/rfc/rfc3281.txt, 2002.

[FK97] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *International Journal of Supercomputer Applications*, 11(2), 1997.

```
[sdecerff@tbn08 test]$ sh select-LFN.sh
Using proxy from /tmp/x509up_u513
Executing program SelectLFN as /NETG/Role=read-test policy: voms-based
 INFO [main] (CRLFileTrustManager.java:120) - Client CN=datagrid.nadc.nl,
 OU=knmi.nl, O=hosts, O=dutchgrid accepted
47 rows returned
---------
LFNOutput
---------
profgdp90619_0614.dat
profgdp90619_0622.dat
profgdp90607_0604.dat
profgdp90607_0612.dat
profgdp90607_0620.dat
profgdp90607_0628.dat
[ more LFNs ]
-end-
[sdecerff@tbn08 test]$
[sdecerff@datagrid ~/client]$
```

Figure 4: Running the program, and its output

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid: Enabling scalable virtual organizations. *International Journal of Supercomputer Applications*, 15(3), 2001.

[HFPS99] R. Housley, W. Ford, W. Polk, and D. Solo. Rfc 2459, internet x.509 public key infrastructure certificate and crl profile. Available on http://www.ietf.org/rfc/rfc2459.txt, 1999.

[MyS01] MySQL AB. Mysql. Available on: http://www.mysql.org, 2001.

[Ogs02] OgsaDai. Open grid services architecture data access and integration. Available on: http://www.ogsadai.org, 2002.

[Pro01] Project Spitfire. Project Spitfire.
Available on: http://spitfire.web.cern.ch, 2001.

[Sec03] Security Coordination Group. Datagrid security design, deliverable 7.6. Technical report, European DataGrid Project, 2003. Available on https://edms.cern.ch/document/344562.