

**CESNET Technical Report 4/2009**

**Fault-tolerant Access Control in Distributed  
Environment – the MetaCentrum  
Authorization Infrastructure**

DANIEL KOUŘIL, MICHAL PROCHÁZKA

Received 8.12.2009

**Abstract**

Although a lot authorization frameworks have emerged recently, they all tend all-or-nothing solutions and thus are hard to integrate with an existing infrastructure. The frameworks also often introduce new critical components, which are too complex and not robust enough, making the deployment and operation difficult. In this report we present an authorization infrastructure, which is simple and robust enough to be used in large distributed environment yet enabling to express and handle a reasonable range of access control policies.

*Keywords:* authorization, grid, distributed environment, push model

## **1 Introduction**

Proper controlling access to resources is an essential part of each environment providing services. Authorization in distributed systems and grids has attracted a lot of attention recently, and many researchers and middleware developers have produced a lot of sophisticated frameworks (for instance Argus<sup>1</sup> or Permis [1]) addressing various aspects of access control.

However, the services developed are quite often very complex in that they utilize XACML [4] or similar XML-based language to express the access control policies. While very general and capable of expressing complex policies, these languages are hard to understand and maintain without additional tools. Likewise, the services introduce new dedicated components, such as Policy Decision Points (PDPs) and Policy Enforcement Points (PEPs) to provide essential functions of an authorization framework. Despite the functionality is needed for each and every authorization system, the architecture with the components decoupled from applications makes it difficult to integrate existing services because significant changes are needed either in the applications or the infrastructure. From the point of view of the deployment, the new components pose additional critical items of the infrastructure, which must be properly monitored and secured in order to ensure they are available anytime because their outage renders the end services not usable.

Outside the grid domain, a lot of applications support authorization based on LDAP [2]. Even though LDAP provides a widely used service with many options, it hardly goes beyond group management as far as authorization is concerned. Each service making use LDAP-based authorization needs to define the policy locally

---

<sup>1</sup> <https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework>

and there is no global view on the policies nor their centralized management available. Relying on LDAP queries also introduces the dependency on a new service, establishing a new single point of failure.

Therefore, despite the variety of the services available we have decided to design a new authorization framework that is easy to integrate with a large distributed environment and does not introduce unnecessary dependency on additional on-line services.

## 2 MetaCentrum

MetaCentrum<sup>2</sup> is a grid infrastructure built and operated by CESNET, which is available to the Czech research community. MetaCentrum provides computation and storage facilities to its users, along with additional services necessary for proper utilization of these resources. Until recently, access control to the MetaCentrum services has not been addressed in an uniform manner. The policies to access individual services were specified on individual basis on particular services. Access to some services was partially managed through the MetaCentrum resource management system to which basic authorization support was added for particular services. Nevertheless, all these individual settings lacked a common approach that would provide a general overview and interfaces to maintain all policies in an uniform way.

However, the growing number of services and increasing focus on services provided to end users has demanded an authorization framework capable of covering the wide range of deployed services and enhance them with sufficient global access control management.

### 2.1 Resource management

MetaCentrum resources are maintained using system Perun [3], which was developed in the course of the MetaCentrum project. Perun takes care of all aspects of resource management, such as user accounts, machine evidence, etc. Each user is identified unambiguously by their MetaCentrum principal name, which may be accompanied by other virtual identities, e.g., the subject name of the user's certificate. Machines can be grouped into *clusters*, which may reflect physical organization or provide a logical grouping of hosts. For instance core machines of MetaCentrum providing essential services are grouped in the *meta-server* virtual cluster so so that the services can be handled in uniform manner.

The Perun server holds the current configuration of all resources as defined by the MetaCentrum administrators and keeps the machines updated. The system was designed for large distributed environments, where various outages occur quite often. Therefore, Perun *pushes* all configuration changes to the machines so that the machines have all configuration settings stored locally, without having to rely on a on-line centralized service. Whenever a change is made in a configuration setting on the Perun server side, the change is evaluated to find out what machines are affected. Then, Perun pushes the changes to the machines using a propagation

---

<sup>2</sup> <http://meta.cesnet.cz/>

protocol. Since the data transferred is sensitive, all the communication is encrypted and mutually authenticated.

All MetaCentrum machines run a *Perun slave daemon* that takes over the messages sent from the server and update the local configuration of the host. For example, commands to create a new Unix account require that changes are done to the appropriate system files (`/etc/passwd`, `/etc/shadow`), the home directory is created and populated with initial user configuration.

## 2.2 Group management

The authorization service is closely tied with Grouper<sup>3</sup> – a system that provides an advanced management of groups of users. Grouper has been developed by Internet2 as a tool to manage institutional and personal groups. We have decided to use Grouper because it provides several features which facilitate management of the groups. Groups can be managed not only by administrators but also by the end users. Grouper strictly manages only groups, all other entities like persons or services are available through connectors from external identity or service management systems. In MetaCentrum, Grouper gets information about users from the resource management system Perun using a direct connection to the Perun's database, therefore Grouper still works with up-to-date records without having to duplicate the records.

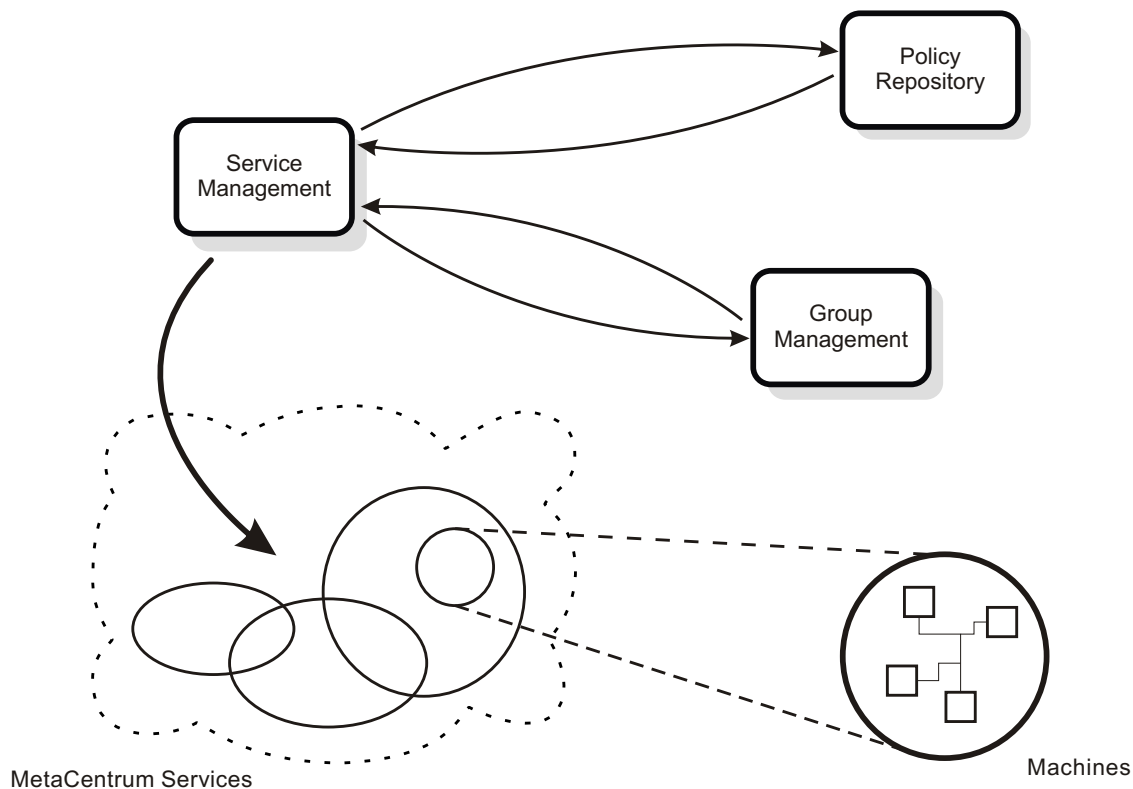
Groups can be stored in either a flat structure or a tree structure where each node is labeled, therefore each a group belongs to a concrete namespace. For the group manipulation Grouper provides a web interface, a web service endpoint and the Grouper shell. Users can manipulate with their groups only within allowed operations scope defined by their roles in the group.

## 3 Authorization in MetaCentrum

The authorization service of MetaCentrum provides a centralized management of access control lists (ACLs) that describe the authorization policy for individual services. Unlike other approaches to authorization these ACLs are not used by the end services directly to make access control decisions or otherwise consulted at the decision time. Instead, the authorization service translates the policy described by the corresponding ACL into configuration snippets that are delivered to the end services so that they can use their native authorization procedures. The update is done upon any change to the access policy. This push model does not introduce any dependency on highly available services.

Given the push model and relying on native authorization support of applications, it is not generally possible to express complex policies such as those depending on actual values of dynamic characteristics, such as the number of free slots in queues or free space on storage systems. Nevertheless, the approach still covers the vast majority of current authorization requirements of MetaCentrum and is easy to integrate to the existing infrastructure since no adaptation to the end services are needed.

<sup>3</sup> <http://grouper.internet2.edu/>



**Figure 1.** Authorization infrastructure

The authorization infrastructure of MetaCentrum consists of several basic components as depicted in Figure 1. The ACL management is the central point to specify and maintain the overall security policy. All the MetaCentrum services managed by the authorization infrastructure are maintained by the service management component. The main goal of this component is to map a service to physical hosts, where the service is available from. The authorization service also ensures that the end service are kept up-to-date in terms of current access control policy. The rest of this section discusses the components of the MetaCentrum authorization infrastructure along with basic requirements on them. Followed that, next section describes in greater details how the components have been implemented.

### 3.1 Policy management

The MetaCentrum authorization service introduces a global policy repository that holds access control policies for all services maintained. Each service's policy is composed of one or more *Policy Entries* assigned to the service:

$$\textit{PolicyEntry} := (\textit{SERVICE}, \textit{ACL})$$

*SERVICE* specifies the resource to which access is controlled. It consists of the service name as given in the resource management and the resource providing the service. The resource can be either a host or a cluster (describing a physical or virtual one):

*SERVICE* := (*service\_name*, *location*)

*ACL* describes the corresponding policy specified for the service:

*ACL* := (*subject*, *type*)

where *subject* specifies either a particular user or an established group of users. The *type* field tells whether the service is *ALLOWED* or *DENIED* for the subject specified. Unlike common perception of ACLs we did not introduce specifications for operations or objects to the format, since they are not needed for the use-cases that we need to address in MetaCentrum. On the other hand, if the requirement for these items emerges in the future, the format is flexible enough to be extended accordingly.

The authorization service is supposed to control access not only to the core MetaCentrum services but also to services created for users that are maintained by them. Therefore, it is foreseen that common users are able to maintain the ACLs of services provided to them by MetaCentrum.

The ACL repository is designed to be closely integrated with the Perun system and Grouper, so that relevant information can be easily exchanged. In particular, whenever a change occurs in an ACL, a notification is immediately dispatched, so that Perun can schedule the update of the corresponding machines and their configuration files. Similarly, when a group used in a ACL changes its membership, similar propagation must be performed as soon as possible.

### 3.2 Service management

The service management is part of the more general resource management provided by Perun. A *service* is an abstraction of any end service provided to the users of MetaCentrum or its staff to which it is needed to control access. The service management component maintains descriptions of each service along with additional information. In order to facilitate authorization it is namely important to identify the physical locations where a particular service is available from, so that configurations of the hosts could be updated properly. The ACLs specify logical locations and may not refer to any physical details, such as particular hostname of a machine or name of a cluster. It is therefore possible to change the physical details about a service without any influence on the access policy specified (e.g., to extend a cluster with a bunch of new nodes or move a service to another physical machine).

The service management is also responsible for pushing any changes to the configurations of the end services. The Perun system already uses the push model for management of users' accounts and other facilities and the very same mechanism is suitable to propagate the authorization information as well. Whenever a change occurs in ACLs, group memberships, and service configurations, an update snippet of configuration reflecting the changes is generated and propagated to all machines affected by the change. The service configuration snippet is generated by a *connector*, which is specified in the service configuration. A connector is a script that for a given service name generates a part of configuration that is acceptable by the end service daemon, and taken into account while doing native authorization decisions. The current ACLs are used as the input to the connector with the groups being ex-

panded to particular usernames of desired types (e.g., Kerberos principal names, X.509 distinguished names, Unix account names). The configuration is delivered to the machines and used by the local configuration management, which updates the services configuration files and restart the service(s) if needed.

### 3.3 Configuration of end services

Each machine that provides a service covered by the authorization service has to be configured and enhanced with the slave component of Perun, which is able to receive the configuration snippets generated by the service management component and update the appropriate local configuration. For instance, an Apache server with the SSL/TLS module makes it possible to specify a list of authorized X.509 DNs that are allowed to access a particular part of the web. The service management component is able to prepare the list given current ACLs and group membership and deliver it to the web server machine. After receiving the new list, the slave part of Perun will override the current one stored at a known place and have the Apache process reloaded by the init script. Obviously, the slave part must be configured according to current Apache configuration, where the list file must be included properly. It is inevitable, therefore, to coordinate with the Apache administrator when the authorization support is being introduced or any changes are applied.

## 4 Implementation of Authorization Infrastructure

This section describes how the authorization infrastructure described in the previous section has been implemented in the MetaCentrum environment.

### 4.1 Implementation of the ACL repository

In order not to increase the number of new components and to ease the maintenance we decided to implement the ACL on the top of Grouper, with the groups providing the back-end to store the ACLs. Each policy entry is represented as a group whose name refers to the resource in question and type while the member describes the subjects that are granted or denied access. The name of a group is of the following format:

*GROUP\_NAME := service\_name;location;location\_type;rule\_type*

Groups added to an ACL are identified by their Grouper name or number. Persons are identified using their primary MetaCentrum identity. Based on requirements of the end service the identity may get translated later on by the connector, in order to provide another identification of the user as needed by the end service. For example SSL/TLS-based services need to know what certificates are assigned to the users, while Kerberos based servers control access based on the users' Kerberos principal names. user.

The authorization mechanism assumes that all changes affecting the access policy are taken into account immediately after they have been applied, which requires that the end services have to be updated as soon as possible. To facilitate the quick reaction without introducing inefficient polling, we make use the Grouper *hooks*.

Grouper enables to define an operation to be invoked upon a particular action, such as changing membership or adding a new group. We have implemented a hook that notifies the Perun service management component on any change of ACLs so a propagation to the end machines can be triggered immediately.

The ACLs are not supposed to be maintained with the Grouper interfaces. Instead, a dedicated Perl API has been developed for ACL management, which hides details about how the ACL are stored in the back-end. The API exports usual functions to create, change, or remove an ACL and to look up the ACL for a given service and location. More information about the API can be found in Appendix A.

A simple web interface utilizing the Perl API has been also developed, see Figure 2. The authentication to the web page is based on the identity federation infrastructure provided by the eduid.cz federation.

The screenshot shows a web browser window titled 'Authorization Service - ACLs - Mozilla Firefox'. The address bar shows 'https://mizar.ics.muni.cz/acl/'. The page title is 'ACL service for MetaCentrum' and it shows 'Logged as kouril@meta.cesnet.cz'. There is a '[Refresh page]' link and a 'Create an ACL' form with fields for Service, Location, Type (HOST/CLUSTER), and a Save button. Below is a table of 'All ACLs'.

Service	Location	Type	Description	Allowed	Denied
X k5login	meta-server	CLUSTER	Pristup na roota pro servisni stroje MetaCentra	<a href="#">Add user</a>   <a href="#">Add group</a> Group metacentrum:security_officers X Group metacentrum:administrators X	<a href="#">Add user</a>   <a href="#">Add group</a>
X k5login	shelob.cesnet.cz	HOST	Pristup na roota pro webovy server MetaCentra	<a href="#">Add user</a>   <a href="#">Add group</a> makub@META X	<a href="#">Add user</a>   <a href="#">Add group</a>
X metaint	meta-internal	CLUSTER	Test	<a href="#">Add user</a>   <a href="#">Add group</a> Group metacentrum:staff X	<a href="#">Add user</a>   <a href="#">Add group</a>
X openvpn	mandor.fi.muni.cz	HOST	Pristup na OpenVPN server MetaCentra	<a href="#">Add user</a>   <a href="#">Add group</a> Group metacentrum:staff X	<a href="#">Add user</a>   <a href="#">Add group</a>
X web	meta-internal	CLUSTER	Interni stranky MetaCentra	<a href="#">Add user</a>   <a href="#">Add group</a> Group metacentrum:staff X	<a href="#">Add user</a>   <a href="#">Add group</a>

© MetaCentrum

**Figure 2.** A screenshot of the ACL web management

Despite the original requirements, the ACL management is deemed as privileged operations that only selected MetaCentrum administrators and services can perform. In order to allow the end users to maintain access policies to their services, they are given ownership of groups included in the ALLOW and DENY rules of the ACLs, instead. That way, the users have a full control over the policy through setting the members of the groups.

## 4.2 Extensions of the resource management system

Perun maintains information about the services in a local database. Each service is assigned a *master* script that is invoked to generate the propagation message. For the service to maintain a users accounts, data to update the passwd files is sent, etc. To maintain the authorization configuration the very same approach was used, since

it fits perfectly the current model and connectors are implemented as the master scripts.

Having utilized the existing Perun propagation, we rely completely on the propagation mechanism. Once invoked, the connectors retrieve the current ACLs for the given service and location and expand the groups to get lists of users that are allowed and denied access. The algorithm pays attention to appropriate evaluation of the policy, taking into account all relevant ACLs. For instance, when propagating changes to a node of a cluster, it is important to evaluate ACLs assigned both to the cluster and the node. The Perun resource management already provides ways to find out all this information, which are used for evaluation. Once all the lists are collected and groups expanded, the DENY lists are subtracted from the ALLOW ones to produce the final list of users that are granted access to a service. The identities of the users are then translated to appropriate representation as required by the end service. Configuration snippets created by the connectors are then transported to the machine through the Perun protocol. The slave script processes the data received storing it to the right place and performing additional steps if needed (see also next section).

Current connectors implement maintenance of the `.k5login` files (used for authorization of Kerberos users accessing ssh and telnet servers), Apache configuration for web pages requiring client certification authentication, and a simple list of certificate subject names used by the OpenVPN server authorization routines.

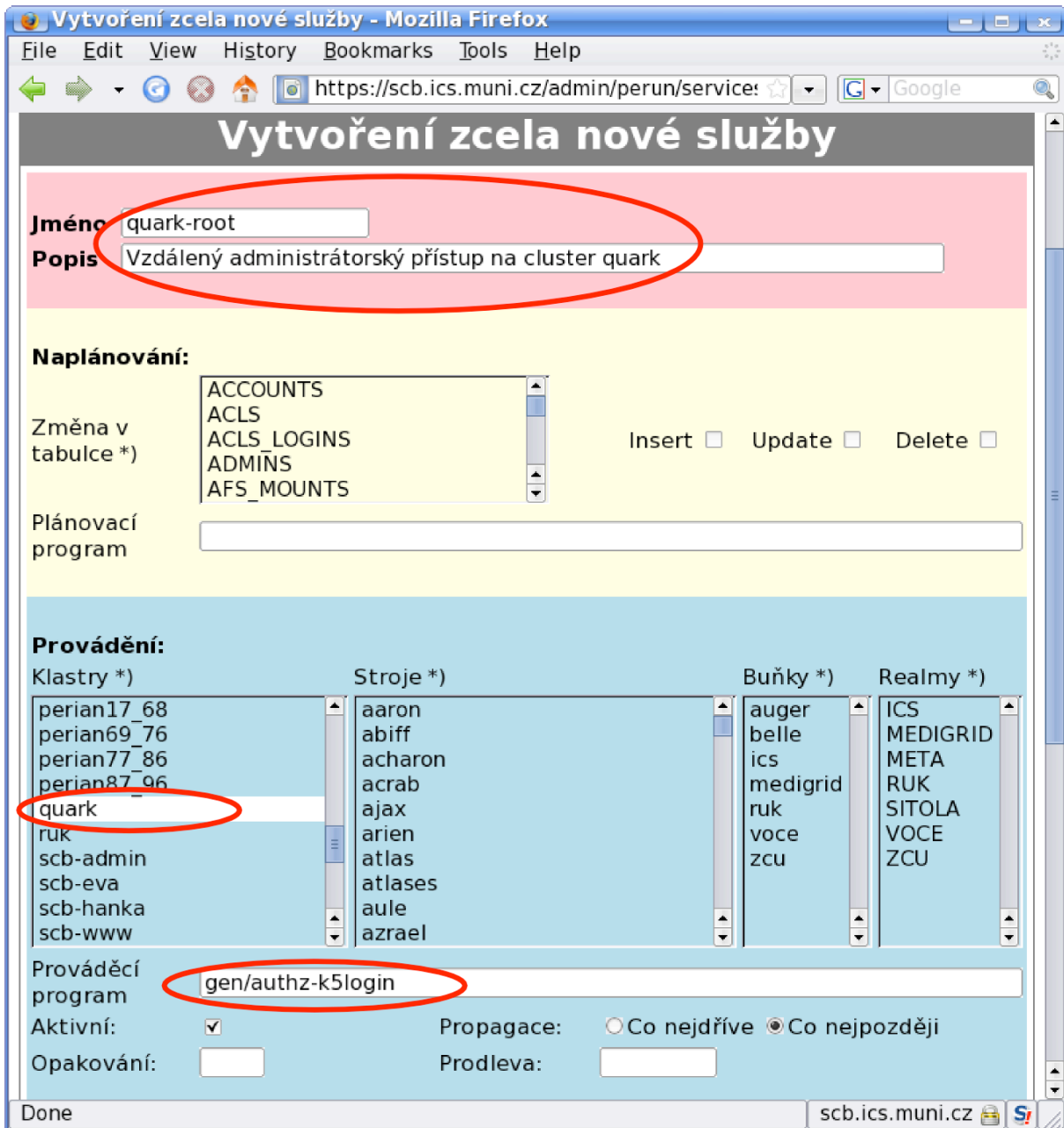
The Perun management web page makes it possible for the administrators to easily define a new service, see Figure 3. Therefore it is easy to add a new service that needs to be covered by MetaCentrum authorization service.

### 4.3 Enabling authorization at end services

Currently, the MetaCentrum authorization infrastructure controls access to several services. Support has been added for managing access to the MetaCentrum VPN server, which shows how users' virtual clusters will be managed once the technology has been made available for the end users. The authorization service is also used to control access to remote login to the root accounts and to the internal web pages available to the MetaCentrum staff. Other services can be easily added.

In order to get an existing service covered by the authorization service a few steps are necessary to perform:

1. Provide the connector for the service that generates the configuration snippet. Several connectors already exists, which may either be reused or provide a guide how a connector can be implemented.
2. Add the service details to the Perun system to describe the new authorization service, assign it to appropriate hosts and/or cluster and define the connector script. The MetaCentrum administration portal can be utilized in this step (see Figure 3).
3. Use the web or Perl API to specify the ACL for the service (see Figure 2).
4. If needed, use Grouper to set memberships in the groups used in the ACLs. The Grouper web page, commands or web service interface can be used for the setting.



**Figure 3.** A screenshot showing introducing a new service

5. Configure the machines. All MetaCentrum machines already support the Perun mechanism so the only thing needed is to enable the connector command and to make sure the slave part of the connector is installed and configured properly.
6. If necessary, apply changes to the service configuration. For instance, make sure that a list of access rules is included in the Apache configuration file.

For services that are maintained by the end-users, the MetaCentrum staff will provide all the steps but the 4th and the appropriate users are granted access to maintain the appropriate groups.

#### 4.4 Access to root accounts – Example of more complex policy

Sometimes it is necessary to provide root access to a MetaCentrum machine to a user who is not a MetaCentrum administrators. For example some network measurements done by students require to tune the setting of the kernel configuration for which root access is necessary. The service is provided only to users with good history and the machine(s) are reinstalled once the experiment is over but still it is highly demanded to have a single point to control the policy. In this example we consider that we provide a node in a cluster that is part of MetaCentrum but also owned and partly maintained by an institution participating in MetaCentrum. Therefore we need to provide three kinds of access, for each a separate service record is needed:

- **root@meta**: a global service covering root access to all the MetaCentrum machines. It includes the smallest group of core MetaCentrum administrators and security staff.
- **root@clustX**: a service describing root access to all nodes of cluster clustX. Usually, it covers the local administrators of the institution that owns the cluster.
- **root@experimentX**: a service established for the experiment for which a set nodes has been dedicated. Users working on the experiment will be granted root access on the nodes.

For each of the services describes a separate set of ACLs can be specified. The groups used in the ACLs are maintained by different users (MetaCentrum security staff, the institution administrators, the user responsible for the experiment). It is flexible enough yet it provides a single place where the current policy is maintained and can be checked. It is the role of the service management and the connectors to combine all the ACLs rules (i.e., both ALLOW and DENY types) to produce a single list of users that is allowed to access the particular nodes. Since the users are required to use a Kerberos authentication to access the services, their Kerberos identities are pushed to the machine and stored in the root's `.k5login` file which is read by the `openssh` and `telnet` daemons.

## 5 Conclusions

The new MetaCentrum authorization infrastructure introduces a possibility to specify access control policies to all MetaCentrum services in a uniform way. The service provides both a web-based GUI and a Perl API for management of the policies, expressed as ACLs. The infrastructure does not contain any new service whose availability is critical for the infrastructure. The end service utilize their native authorization mechanism based solely on local configuration, which is maintained from a central place and changes upon changes. Since no adaptations of the applications are needed, the authorization service was easy to deploy in the existing MetaCentrum environment.

## 6 Acknowledgments

This MetaCentrum project and the work described in this report have been supported by research intent “Optical Network of National Research and Its New Applications” (MŠM 6383917201).

## References

- [1] CHADWICK, D. W.; ZHAO, G.; OTENKO, S.; LABORDE, R.; SU, L.; NGUYEN, T. A. PERMIS: a modular authorization infrastructure. *Concurrency and Computation: Practice and Experience*. 2008, vol. 20, no. 11, p. 1341–1357. Online ISSN: 1532-0634.
- [2] HODGES, J.; MORGAN, R. *Lightweight Directory Access Protocol (v3): Technical Specification*. RFC 3377<sup>4</sup>, IETF, September 2002.
- [3] KŘENEK, A.; SEBASTIANOVÁ, Z.; SITERA, J. *Perun – systém pro řízení přístupu uživatelů k prostředkům METACentra [Perun—User Access Control System for METACentrum Resources]*. Technical Report 1/2004<sup>5</sup>. Praha: CESNET, 2004. In Czech.
- [4] OASIS. *XACML 2.0 Core: eXtensible Access Control Markup Language (XACML) Version 2.0*. October 2005.

## Appendix A. Perl API to Manage ACLs

Two Perl packages have been developed to facilitate the management of ACLs stored in the MetaCentrum policy repository. As mentioned, Grouper is used as the repository back-end to store the ACLs and the API utilizes the web service interface of Grouper sent over an https channel in order to perform the necessary operation with the groups.

### A.1 Basic routines to manipulate ACLs – Acl.pm

The Policy Entry describing a single access control rule (see Section 3.1) is defined as a hash:

```
%service = ('service' => <service_name>,
            'location' => {'name' => <destination_name>,
                          'type' => <type>}),
```

where <type> can be one of CLUSTER and HOST.

Similarly, a subject for the ACLs is modeled as hash:

```
%subject = ('name' => <subject_name>, 'type' => <type>),
```

<sup>4</sup> <http://tools.ietf.org/html/rfc3377.html>

<sup>5</sup> <http://www.cesnet.cz/doc/techzpravy/2004/perun-2.2/perun-2.2.pdf>

where <type> is one of GROUP or USER. The GROUP type denotes a Grouper group name, while USER refers to an identifier of a particular user. In the latter case, a common MetaCentrum identification is the user's username assigned.

The types introduced are handled by several functions and methods.

`new(%service)` \creates a new object referring to the service given in the parameters.

`allow(%subject)`, `deny(%subject)` \methods to specify a subject that is allowed or denied, respectively.

`unallow(%subject)`, `undeny(%subject)` \methods to revoke previous setting  
`subjects()` \returns a list of all subjects that are currently defined. The results is a hash containing both allowed and denied subjects:

```
%subjects = (ALLOW => <@list_allow>, DENY => <@list_deny>),
```

where both the lists are composed of the %subject hashes.

### A.1.1 Example

The following code snippet shows how a simple policy can be defined. In this example a root access to the skirit cluster is being specified, allowing all the members of the meta:admin group but kouril@META to access the root account at cluster skirit.

```
$service = Acl->new('service' => 'root',
                  'location' => {'name' => 'skirit' });
$service->allow('name' => 'meta:admins');
$service->deny('name' => 'kouril@META', 'type' => Acl::USER);
```

## A.1 Auxiliary routines – AclUtils.pm

The most important routine of this package is `extractAllowedSubjects`. Given a list of ACLs the function computes the list of principal names that are allowed in all the ACLs, i.e., it expands all the group names and ensure that all DENIED principals are omitted from the final list.

### A.1.1 Example

```
$clust = Acl->new('service' => 'root',
                'location' => {'name' => 'skirit'});
$host = Acl->new('service' => 'root',
                'location' => {'name' => 'skirit1', 'type' =>
Acl::HOST});
$allowed = extractAllowedSubjects($clust, $host);
```