Middleware Security

in Selected Grid Infrastructures (2010)

BiG Grid
the dutch e-science grid

eGEE
Enabling Grids for E-sciencE

NIKHEF pdp

David Groep, Nikhef
with graphics by many others from publicly available sources...
Grid Security Middleware mechanisms for protecting the e-Infrastructure
V What to expect?

What might be covered

> How to deal with AuthN
> AuthZ frameworks
> Access control in services
> Unix credential mapping
> Pilot jobs and late binding
> Security interoperability
> Storage access control
> Data Security and Privacy

What will not be covered

> How to write secure code
  > Look at http://pages.cs.wisc.edu/~kupsch/
> Current vulnerabilities
  > They’re secret for a reason...
> Most of the federation work
  > Milan will tell you all
> The latest WS-* *ML specs

> ... with a slight EGEE & C + Unix bias, sorry ...
A taxonomy of this middleware talk

- To the Unix Domain
  - ACLs and banning
- Compute Services
  - glexec
  - Centralizing access control
  - Late Job Binding
  - Long-running job renewal
- Middleware Authorization Frameworks
- Authentication and Identity Credentials
- Community services
- Community organisation
- Encrypted storage
- Storage models

… with a slight EGEE & C + Unix bias, sorry …
Trust infrastructures and PKI: verifying authenticity
Delegation and proxies
Getting practical about failures

SECURITY MECHANISM
FOUNDATIONS AND SCOPE
Elements of Trust

> Authentication
  > Who are you?
  > Who says so?
  > How long ago was that statement made?
  > Have you changed since then?

> Authorization
  > Why should I let you in?
  > What are you allowed to do?
  > By whom? *Who* said you could do that?
  > And how long ago was *that* statement made?
V Authentication models

> Direct user-to-site
  > passwords, enterprise PKI, Kerberos

> PKI with trusted third parties

> Federated access
  > Controlled & policy based
  > Open or bi-lateral, e.g., OpenID

> Identity meta-system
  > Infocard type systems: will they materialize?
X.509: add identifiers to a public key

> Authentic binding between
  > Subject name
  > A public key
  > A validity period
  > Zero or more extensions
  > ... that can contain identifiers
  > ... or policies

> Signed by an issuer
  > Yourself: self-signed cert
  > Trusted third party, ‘CA’
V Verification steps

> Check signature chain up to a trusted root
  > OpenSSL (thus most middleware) root of trust *must* be self-signed
  > Trust anchors
    • ‘.0’ files in ‘PEM’ format, e.g. from IGTF as RPM, tgz or JKS
  > Revocation
    • Lists ‘.r0’ files in PEM format, retrieved by tools: fetch-crl
    • OSCP not operationally deployed in grids

> Check extensions
  > basicConstraints and keyUsage, but others must be ‘sane’ as well
  > ‘interesting’ errors in case other extensions are wrong, beware!

> Check RP namespace constraints
Signing policy files

> Constrain name space to specified subject names

<table>
<thead>
<tr>
<th>access_id_CA</th>
<th>X509 access</th>
<th>'/C=NL/O=NIKHEF/CN=NIKHEF medium-security certification auth'</th>
</tr>
</thead>
<tbody>
<tr>
<td>pos_rights</td>
<td>globus</td>
<td>CA:sign</td>
</tr>
<tr>
<td>cond_subjects</td>
<td>globus</td>
<td>'/C=NL/O=NIKHEF/CN=NIKHEF medium-security certification auth'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'/O=dutchgrid/O=users/*'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'/O=dutchgrid/O=hosts/*'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>'/O=dutchgrid/O=robots/*'</td>
</tr>
</tbody>
</table>

> For now, specific to Grids

> Recognised in

- Globus Toolkit C core, and Java in 4.2+
- gLite Trust Manager
- GridSite (recent versions only)

> Parsing is prone to many, many bugs!

> See OGF CAOPS-WG “RPDNC Policies” document
Building up: CA hierarchies

Not all paths need to be equally trusted by a relying party - i.e. a site, a user or a VO
Hierarchies in middleware

Namespace constrains aid in securing hierarchies

```plaintext
# EACL - AAACertificateServices
#
access_id_CA   X509  '/C=GB/ST=Greater Manchester/L=Salford/O=Comodo CA Limited/CN=AAA
Certificate Services'
pos_rights    globus CA:sign
cond_subjects globus "'/C=GB/ST=Greater Manchester/L=Salford/O=Comodo CA Limited/CN=AAA
Certificate Services" "'/C=US/ST=UT/L=Salt Lake City/O=The USERTRUST Network/OU=http://
www.usertrust.com/CN=UTN-USERFirst-Client Authentication and Email"

# EACL - UTNAAAClient
#
access_id_CA   X509  '/C=US/ST=UT/L=Salt Lake City/O=The USERTRUST Network/OU=http://
www.usertrust.com/CN=UTN-USERFirst-Client Authentication and Email'
pos_rights    globus CA:sign
cond_subjects globus "'/C=NL/O=TERENA/CN=TERENA eScience Personal CA"

# EACL - TERENAAeSciencePersonalCA
#
access_id_CA   X509  '/C=NL/O=TERENA/CN=TERENA eScience Personal CA'
pos_rights    globus CA:sign
cond_subjects globus "'/DC=org/DC=terena/DC=tcs/*"
```
Hierarchies in middleware

> Alternate ‘.namespaces’ format (e.g. VOMS)

```plaintext
#CA Hierarchy anchored at AAACertificateServices for
# the TCS eScience Personal CA

TO Issuer "/C=GB/ST=Greater Manchester/L=Salford/O=Comodo CA Limited/CN=AAA Certificate Services"
   PERMIT Subject "/C=US/ST=UT/L=Salt Lake City/O=The USERTRUST Network/OU=http://www.usertrust.com/CN=UTN-USERFirst-Client Authentication and Email"

TO Issuer "/C=US/ST=UT/L=Salt Lake City/O=The USERTRUST Network/OU=http://www.usertrust.com/CN=UTN-USERFirst-Client Authentication and Email"
   PERMIT Subject "/C=NL/O=TERENA/CN=TERENA eScience Personal CA"

TO Issuer "/C=NL/O=TERENA/CN=TERENA eScience Personal CA"
   PERMIT Subject "/DC=org/DC=terena/DC=tcs/*"
```
Delegation – why break the recursion?

- Mechanism to have someone, or something
  - a program – act on your behalf
    - as yourself
    - with a (sub)set of your rights
- Essential for the grid model to work

- GSI/PKI and recent SAML drafts define this
  - GSI (PKI) through ‘proxy’ certificates (see RFC3820)
  - SAML through Subject Confirmation,
    (linking to at least one key or name)
Daisy-chaining proxy delegation

1. Authenticate and DELEGATE RIGHTS

Resource Broker submission and job management host

2. Authenticate and DELEGATE RIGHTS

3. Authenticate

User Job
Data processing, reading and writing remote files
V Delegation, but to whom?

> ‘normal’ proxies form a chain

  > Subject name of the proxy derived from issuer

    “/DC=org/DC=example/CN=John Doe/CN=24623/CN=535431”

    is likely a proxy for user

    “/DC=org/DC=example/CN=John Doe”

  > May contain path-length constraint

  > May contain policy constraints

  > And: legacy (pre-3820) proxies abound

> **But: use the name of the real end-entity for authZ!**

Note that

> in SAML, delegation can be to any NameID

> in RFC3820 these are called ‘independent proxies’
Verifying authentication and X.509

> ‘Conventional’ PKI
  > OpenSSL, Apache mod_ssl
  > Java JCE providers, such as BouncyCastle
  > Perl, Python usually wrappers around OpenSSL

> With proxy support
  > OpenSSL (but beware of outstanding issues!)
  > Globus Toolkit (C, Java)
  > GridSite
  > ProxyVerify library
  > TrustManager

> Always ensure the proxy policies are implemented
Verification middleware options

> Plain OpenSSL (C)
  > On its own, it is *not enough* to verify a credential
  > Need to add your validation routines for (proxy) credentials
e.g. [http://www.nikhef.nl/~janjust/proxy-verify/](http://www.nikhef.nl/~janjust/proxy-verify/)
  > No single library available, though – re-implemented many times

> GridSite (Andre McNab, C)

> Globus Toolkit GSI utils (C) and Java tools

> gLite Trust Manager (Java)
  > based on BouncyCastle

Most of this is based on OpenSSL or BouncyCastle, but ...
V Trust anchor formats are diversifying

> Java based software (e.g. Unicore)
  > Java Key Store
  > Supports keys over 2048 bits only since Java 1.4

> NSS
  > Increasingly deployed as alternative to OpenSSL
  > E.g. by RedHat and its distribution of apache mod_nss
  > Binary trust store format (‘key3.db’, ‘cert8.db’)
  > Trust anchors can be qualified for purposes
  > But not a ‘pluggable’ format ...

> OpenSSL
  > Changed its format in v1 for no apparent reason
AuthN and AuthZ, a User View

Potential problems in AuthN and AuthZ

- Trust anchors inconsistent on client and server
- Certificates revoked
- CRL outdated
- Time on client and server is different
- Proxy has expired or has wrong attributes
- User not member of the (proper) VO
- ...

- Error messages usually incomprehensible
- Which side it ‘at fault’ is unclear
V In Real Life …

Error creating PKCS#7 structure
1688:error:0B080074:x509 certificate routines:X509_check_private_key:key values mismatch:x509_cmp.c:411:
1688:error:2107407F:PKCS7 routines:PKCS7_sign:private key does not match certificate:pk7_smime.c:76:

End of file reached

Error -12227

> For testing the connection and interpreting the diagnostics try, e.g., the connection verify utility

> http://www.nikhef.nl/grid/client-connect
USER COMMUNITY MODELS
Authorization: VO representations

- VO is a directory (database) with members, groups, roles
- based on identifiers issued at the AuthN stage
- Membership information is to be conveyed to the resource providers
  
  - configured statically, out of band
  - in advance, by periodically pulling lists
    VO (LDAP) directories
  - in VO-signed assertions pushed with the request: VOMS, Community AuthZ Service
  - Push or pull assertions via SAML
VO LDAP model
VOMS: X.509 as a container

Virtual Organisation Management System (VOMS)

- developed by INFN for EU DataTAG and EGEE
- used by VOs in EGEE, Open Science Grid, NAREGI, ...
- push-model signed VO membership tokens
  - using the traditional X.509 ‘proxy’ certificate for trans-shipment
  - fully backward-compatible with only-identity-based mechanisms

VOMS proxy with embedded VO assertion

<table>
<thead>
<tr>
<th>Attribute Certificate</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTEGER</td>
</tr>
<tr>
<td>SUBJECT</td>
</tr>
<tr>
<td>SERIAL</td>
</tr>
<tr>
<td>ISSUER</td>
</tr>
<tr>
<td>OCTET STRING</td>
</tr>
<tr>
<td>OCTET STRING</td>
</tr>
<tr>
<td>OBJECT</td>
</tr>
<tr>
<td>AuthorityIdentifier</td>
</tr>
<tr>
<td>SignatureAlgorithm</td>
</tr>
</tbody>
</table>
V VOMS model

Identity Certificate
/O=dutchgrid/O=users/O=nikhef/CN=Wouter

AuthN Authority/CA

Resource Provider

VO member (scientist)
Registration (once)

VO Administrator

prove identity to VOMS server

VO membership assertion

VOMS proxy

sent by the VO admin via trusted means

verify

‘VOMS’ assertion embedded in proxy conveyed to resource

Grant access to resource and optionally do rights, Unix account, mappings

Identity Certificate
/O=CH/O=CERN/CN=lxbr2341.cern.ch

International Symposium on Grid Computing
April 2009
V  GUMS model

> VO configuration replicated locally at the site
> Here, pushed VOMS attributes are advisory only
Towards a multi-authority world (AAI)

Interlinking of technologies can be done at various points:

1. Authentication: linking (federations of) identity providers to the existing grid AuthN systems
   - ‘Short-Lived Credential Services’ translation bridges
2. Populate VO databases with UHO Attributes
3. Equip resource providers to also inspect UHO attributes
4. Expressing VO attributes as function of UHO attributes
   - and most probably many other options as well ...

Leads to assertions with multiple LoAs in the same decision:
- thus all assertions should carry their LoA
- expressed in a way that’s recognisable
- and the LoA attested to by ‘third parties’ (i.e. the federation)
Federations

- A common Authentication and Authorization Infrastructure
- Allow access to common resources with a single credential
V A Federated Grid CA

> Use your federation ID
> ... to authenticate to a service
> ... that issues a certificate
> ... recognised by the Grid today

Implementations:
- SWITCHaai SLCS
- TERENA Grid CA Service

Outdated Graphic from: Jan Meijer, UNINETT

---

International Symposium on Grid Computing

April 2009
Attributes from multi-authority world

- In ‘conventional’ grids, all attributes assigned by VO
- But there are many more attributes

- VASH: ‘VOMS Attributes from Shibboleth’
  - Populate VOMS with generic attributes
  - Part of gLite (SWITCH)

http://www.switch.ch/grid/vash/

Graphic: Christoph Witzig, SWITCH
Putting home attributes in the VO

> Characteristics
> The VO will know the source of the attributes
> Resource can make a decision on combined VO and UHO attributes
> but for the outside world, the VO now has asserted to the validity of the UHO attributes – over which the VO has hardly any control
V Attribute collection ‘at the resource’

Characteristics

- The RP (at the decision point) knows the source of all attributes
- but has to combine these and make the ‘informed decision’
- is suddenly faced with a decision on quality from different assertions
- needs to push a kind of ‘session identifier’ to select a role at the target resource

Graphic: the GridShib project (NCSA)
http://gridshib.globus.org/docs/gridshib/deploy-scenarios.html

Chistoph Witzig, SWITCH, GGF16, February 2006
Container versus service level
Logical authZ structure: PEP,PDP,PAP,PEP
Frameworks
AUTHORIZATION FRAMEWORKS
V A multi-authority world

> Authorization elements (from OGSA 1.0)
Logical Elements in authorization

1. Access Request
   - PEP (Policy Enforcement Point)
     - 8 Obligations
   - Obligation service

2. Request 7. Response
   - PDP (Policy Decision Point)
     - 4. Attribute query
     - 6. Attribute

3. Policy
   - PAP (Policy Access Point)
   - Subject
   - Resource
   - Environment

4. Attribute query
   - PIP (Policy Information Point)
     - 5a. Subject attributes
     - 5b. Resource attributes
     - 5c. Environment attributes

― beware that translating architecture to implementation 1:1 is a recipe for disaster ―
V Control points

**Container based**
- Single control point
- Agnostic to service semantics

**Service based**
- Many control points
- Authorization can depend on requested action and resource
V Frameworks

> (chain of) decision making modules controlling access
> Loosely or tightly coupled to a service or container
> Generic ‘library’, or tied into the service business logic

example: GT4/Java
V Some framework implementations

- PRIMA-SAZ-GUMS-gPlazma suite
- Globus Toolkit v4 Authorization Framework
- Site Access Control ‘LCAS-LCMAPS’ suite
- Argus (gLite)
- GridSite & GACL
- ...

... and don’t forget ‘native’ service implementations
Different frameworks

> Each framework has
  > own calling semantics (but may interoperate at the back)
  > its own form of logging and auditing

> Most provide
  > Validity checking of credentials
  > Access control based on Subject DN and VOMS FQANs
  > Subject DN banning capability

> And some have specific features, e.g.,
  > Capability to process arbitrary ‘XACML’ (composite) policies
  > Calling out to obtain new user attributes
  > Limiting the user executables, or proxy life time, ...
Example: running compute jobs
Access control: gatekeepers, gLEexec, ban lists, and GACL

ACCESS CONTROL FOR COMPUTE
Job Submission Today

User submits his jobs to a resource through a ‘cloud’ of intermediaries

Direct binding of payload and submitted grid job
- job contains all the user’s business
- access control is done at the site’s edge
- inside the site, the user job has a specific, site-local, system identity
V Access Control for Compute on Unix

> System access needing assignment of Unix account
  > Either locally on the node (grid-mapfile, LCMAPS)
  > or through call-outs to GUMS (Prima), Argus PEP-C client, or SCAS
Example: LCAS in basic authorization

> Pluggable authorization framework in C

  > Independent modules (‘shared objects’) called based on simple ‘boolean-AND’ policy description

> Decisions based on

  > Allowed user or VOMS FQAN list
  > Deny based on a separate ‘ban’ list with wildcards
  > GACL policy
  > Allowed-executable (‘RSL’ matching)
  > Time slots
  > L&B2-policy module

http://www.nikhef.nl/grid/lcaslcmaps/
V LCAS example

```
# @(#) lcas.db
pluginname=lcas_userban.mod,pluginargs=ban_users.db
pluginname=lcas_voms.mod,pluginargs="-vomsdir/etc/grid-security/vomsdir/ ..."
```

```
# @(#) ban_users.db
/DC=org/DC=example/CN=Sherlock Holmes
/DC=gov/DC=somelab/OU=CDF/CN=* 
```

Only DN c.q. FQAN used from ... /etc/grid-security/grid-mapfile

```
"/O=dutchgrid/O=users/O=nikhef/CN=David Groep" .pvier
"/O=dutchgrid/O=users/O=nikhef/CN=Oscar Koeroo" okoeroo "/C=AT/O=AustrianGrid/OU=UIBK/OU=OrgUnit/CN=Name Suppressed" .esr
"/vlemed/Role=NULL/Capability=NULL" .vlemed
"/vlemed" .vlemed
"/vo.gear.cern.ch/Role=NULL/Capability=NULL" .poola
"/vo.gear.cern.ch" .poola
"/vo.gear.cern.ch/Role=lcgadmin/Capability=NULL" .troi
"/vo.gear.cern.ch/Role=lcgadmin" .troi
```
V Argus Policies and banning

> Integrated policy with distributed mechanism

```
resource ".*" {
  obligation "http://glite.org/xacml/obligation/local-environment-map" {}
  action ".*" {
    rule deny {
      subject = "CN=Alberto Forti,L=CNAF,OU=Personal Certificate,O=INFN,C=IT"
    }
    rule deny {
      fqan = /dteam/test
    }
    rule deny {
      pfqan = "/lsgrid/Role=pilot"
    }
    rule permit {
      vo = "lsgrid"
    }
  }
}
```

> ‘pushed’ policies can implement central banning

https://twiki.cern.ch/twiki/bin/view/EGEE/SimplifiedPolicyLanguage
https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework
gLite WMS access control: GACL

```
<gacl version="0.0.1">
  <entry>
    <voms>
      <fqan>lofar/ROLE=admin</fqan>
    </voms>
    <allow><exec/></allow>
  </entry>

  <entry>
    <voms>
      <fqan>lsgrid</fqan>
    </voms>
    <allow><exec/></allow>
  </entry>

  <entry>
    <person>
      <dn>/DC=org/DC=example/O=HEP/O=PKU/OU=PHYS/CN=Some Person</dn>
    </person>
    <deny><exec/></deny>
  </entry>
</gacl>
```

GridSite and LCAS can do GACL as well, though ...
V  GUMS access control

GUMS is a central-service only mapping service
> Database with a ‘site’ dump of the VO membership
> Tools to manipulate that database
> e.g. banning a user or a VO

https://twiki.grid.iu.edu/bin/view/Security/GUMS--DevelopmentandAdditions

# an individual that is not a VO member
/DC=org/DC=doegrids/OU=People/CN=Jay Packard 335585,

# an individual from any VO
/DC=org/DC=doegrids/OU=People/CN=Jay Packard 335585, .*

# or an individual from the Atlas production role
/DC=org/DC=doegrids/OU=People/CN=Jay Packard 335585, //atlas/usatlas/Role=production.*

> please hold for a central service based on LCAS-LCMAPS...
**But notably different**

- **gLite WMS**
  - Uses GACL libraries directly and exclusively

- **Storage access control, e.g. DPM**
  - Has built-in native handing of groups via POSIX ACLs expressed as VOMS FQANs

- **Native GRAM, GSIssh, and GridFTP in GT <= 5.0**
  - Has only a static DN map file
  - Unless configured to use LCAS-LCMAPS or PRIMA-GUMS

- ...
But basic yes-no does not get you far

> *If yes, what are you allowed to do?*

> Credential mapping via obligations, e.g. unix account, to limit what a user can do and disambiguate users

> Intended side effects: allocating or creating accounts ... or virtual machines, or ...

> Limit access to specific (batch) queues, or specific systems

> Additional software needed

> Handling ‘obligations’ conveyed with a decision

> LCMAPS: account mappings, AFS tokens

> Argus: pluggable obligation handlers per application
  
  • E.g. used by LCMAPS again when talking to an Argus service
Credential mapping
Running jobs

*Long-running jobs and MyProxy*

*Addressing late-binding with gLEexec*

**TO THE UNIX WORLD**
Computing jobs in a multi-user Unix site

Site Access Control
(LCAS/LCMAPS)

Job Manager

Job Manager

Gatekeeper
supervisor

Library invocation in the gatekeeper
or invoked by gLExec in CREAM

Worker Node

User Job
Unix uid specific to User

Worker Node

User Job
Unix uid specific to User

Worker Node

User Job
Unix uid specific to User

Site Boundary
To the Unix world: Problem

Unix does not talk Grid, so translation is needed between grid and local identity

1. this translation has to happen somewhere
2. something needs to do that
To the Unix world: LCMAPS

Two things need to happen

> Figure out which account to use
  > Acquisition
    collect attributes and obligations
    allocate or make an account
    obtain a mapping from a service

> Make sure you get there
  > Enforcement
    modify accounts if needed (LDAP)
    obtain AFS tokens for file access
  > change effective user id of process
    needs to be the last step

run as root
credential: …/CN=Pietje Puk

run as target user
uid: ppuk001
uidNumber: 96201
V LCMAPS modules

> Acquisition
(voms)local{account,group}, (voms)pool{account,group}, GUMS, verify-proxy, scas-client

> Enforcement
posix_enf, ldap_enf, afs, jobRepository

http://www.nikhef.nl/grid/lcaslcmaps/
V LCMAPS configuration example (local)

```
# LCMAPS config file for glexec generated by YAIM

vomslocalgroup = "lcmaps_voms_localgroup.mod ...")
vomslocalaccount = "lcmaps_voms_localaccount.mod ...")
vomspoolaccount = "lcmaps_voms_poolaccount.mod ...")
localaccount = "lcmaps_localaccount.mod"
    " -gridmapfile /etc/grid-security/grid-mapfile"
poolaccount = "lcmaps_poolaccount.mod"
    " -override_inconsistency"
    " -gridmapfile /etc/grid-security/grid-mapfile"
    " -gridmapdir /share/gridmapdir"
good = "lcmaps_dummy_good.mod"

# Policies: DN-local -> VO-static -> VO-pool -> DN-pool
static_account_mapping:
localaccount -> good

voms_mapping:
vomslocalgroup -> vomslocalaccount
vomslocalaccount -> good | vomspoolaccount

classic_poolaccount:
poolaccount -> good
```

Policy sequence depends on the service!
Mapping, but where

> Locally at the service end (the CE node)
  > LCMAPS
  > Globus ‘authz call-out’ loaded with LCMAPS
  > Classic ‘gss_assist’ grid-mapfile

> At a (central) mapping/authz service
  > PRIMA + GUS
  > LCMAPS + SCAS
  > LCMAPS + Argus
  > gPlazma + GUMS (some forms of storage)
  > GT call-out talking to LCMAPS or Argus
V

Pilot jobs
Impact on sites

LATE BINDING
Classic job submission models

> In the submission models shown, submission of the user job to the batch system is done with the *original job owner’s* mapped (uid, gid) identity.

> grid-to-local identity mapping is done *only* on the front-end system (CE).
  > batch system accounting provides per-user records
  > inspection shows Unix process on worker nodes and in batch queue per-user
Late binding: pilot jobs

Job submission gets more and more intricate ...

- Late binding of jobs to job slots via pilot jobs
  - some users and communities develop and prefer to use proprietary, VO-specific, scheduling & job management
    - ‘visible’ job is a pilot: a small placeholder that downloads a real job
    - first establishing an overlay network,
    - subsequent scheduling and starting of jobs is faster

- it is not committed to any particular task on launch
- perhaps not even bound to a particular user!

- this scheduling is orthogonal to the site-provided systems
Every user a pilot
V Pilot job incentives

Some Pros:

> Worker node validation and matching to task properties
> Intra-VO priorities can be reshuffled on the fly without involving site administrators
> Avoid jobs sitting in queues when they could run elsewhere

*From: https://wlcg-tf.hep.ac.uk/wiki/Multi_User_Pilot_Jobs*

> For any kind of pilot job:
  > Frameworks such as Condor glide-in, DIRAC, PANDA, ... or Topos, are popular, because they are ‘easy’ (that’s why there are so many of them!)
  > Single-user pilot jobs are no different than other jobs when you allow network connections to and from the WNs
  > Of course:
    any framework used to distribute payload gives additional attack surface
Multi-user pilot jobs

1. All pilot jobs are submitted by a single (or a few) individuals from a user community (VO)
   > Creating an overlay network of waiting pilot jobs

2. VO maintains a task queue to which people (presumably from the VO) can submit their work

3. Users put their programs up on the task queue

4. Pilot jobs on the worker node looks for work from that task queue to get its payload

5. Pilot jobs can execute work for one or more users in sequence, until wall time is consumed
VO overlay networks: MUPJ
A resource view of MUPJs

Multi-user pilot jobs hiding in the classic model

Classic model
Pros and Cons of MUpilot jobs

In current ‘you only see the VO pilot submitter’ model:

> Loss of control over scheduling/workload assignment, e.g.
  > site admin cannot adjust share of specific user overloading e.g. the Storage Element (only the pilots are seen by the batch system) and might need to:
  > ban entire VO instead of user from the SE and/or CE, or
  > reduce the entire VO share
  > Is that acceptable in case of a non-confirmed incident?

> Traceability and incident handling issues

Advantages

> you only see & need to configure a single user
> It’s not complicated, and no software/config is needed

Extensive summary of technical issues (pros and cons):
https://wlcg-tf.hep.ac.uk/wiki/Multi_User_Pilot_Jobs
Traceability and compromises

> Post-factum: in case of security incidents:
  > Complete & confirmed compromise is simple: ban VO
  > In case of suspicion: to ban or not to ban, that’s the question
    • There is no ‘commensurate’ way to contain compromises
    • Do you know which users are inside the VO?
      No: the list is largely private
      No: it takes a while for a VO to respond to ‘is this user known’?
      No: the VO will ban user only in case they think (s)he is malicious – that may be different from your view, or from the AIVD’s view, or ... 
    • So: the VO may or may not block
    • The site is left in the cold: there is no ‘easy’ way out except blocking the entire VO, which then likely is not ‘acceptable’
V Traceability and compromises

> Protecting user payload, other users, and the pilot framework itself from malicious payloads

> To some extent a problem for the VO framework, not for the site
> Not clear which payload caused the problem: all of them are suspect
> User proxies (when used) can be stolen by rogue payloads
> ... or the proxy of the pilot job submitter itself can be stolen
> Risk for other user to be held legally accountable
> Cross-infection of users by modifying key scripts and environment of the framework users at each site

> Helps admins understand which user is causing a problem
V Traceability and compromises

> Ante-factum requirements
Sites may need proof of the identity of who was
(or is about to!) use the resources at any time,
in particular the identities involved in any ongoing incidents

> Information supplied by the VO may be (legally) insufficient or too late

> Privacy laws might hamper the flow of such information back and forth
  > c.f. the German government’s censorship bill, with the list of domains that
    a DNS server must block, but which cannot be published by the enforcing ISP
  > Or other government requirements or ‘requests’ that need to be cloaked
V MUPJ security issues

With multi users use a common pilot job deployment Users, by design, will use the same account at the site

> Accountability
  no longer clear at the site who is responsible for activity

> Integrity
  a compromise of any user using the MUPJ framework ‘compromises’ the entire framework

the framework can’t protect itself against such compromise unless you allow change of system uid/gid

> Site access control policies are ignored

> ... and several more ...
V

Policy
gLExec
Cooperative control

RECOVERING CONTROL
Recovering control: policy

- Policy itself
  - E.g. [https://edms.cern.ch/document/855383](https://edms.cern.ch/document/855383)

- Collaboration with the VOs and frameworks
  *You cannot do without them!*
  - Vulnerability assessment of the framework software
  - Work jointly to implement and honour controls
  - Where relevant: ‘trust, but verify’

- Provide middleware control mechanisms
  - Supporting site requirements on honouring policy
  - Support Vos in maintaining framework integrity
  - Protect against ‘unfortunate’ user mistakes
V Recovery control: mechanisms

1. Unix-level sandboxing
   > POSIX user-id and group-id mechanisms for protection
     > Enforced by the ‘job accepting elements’:
       • Gatekeeper in EGEE (Globus and lcg-CE), TeraGrid and selected
         HPC sites
       • Unicore TSI
       • gLite CREAM-CE via sudo

2. VM sandboxing
   > Not widely available yet
Pushing access control downwards

Making multi-user pilot jobs explicit with distributed Site Access Control (SAC)

- on a cooperative basis -

[Diagram of multi-user pilot job management]
Recovering Control

1. Make pilot job subject to normal site policies for jobs
   - VO submits a pilot job to the batch system
     - the VO ‘pilot job’ submitter is responsible for the pilot behaviour
       - *this might be a specific role in the VO, or a locally registered ‘special’ user at each site*
     - Pilot job obtains the true user job, and presents the user credentials and the job (executable name) to the site (glexec) to request a decision on a cooperative basis

2. Preventing ‘back-manipulation’ of the pilot job
   - make sure user workload cannot manipulate the pilot
   - project sensitive data in the pilot environment (proxy!)

   - by changing uid for target workload away from the pilot
Recovering control: gLExec

Virtual Organisation

User Job

VO Workload Management System or Job Queue

User Job

Site Boundary

Grid Computing Service
Site or VO CE or traditional gatekeeper mechanism

LRMS Queue

VO Pilot Job (VO uid)

Worker Node

VO Pilot Job (VO uid)

gLexec

User Job Unix uid specific to User

Grid Workload Management Systems

Pilot Job
What is gLExec?

gLExec

a thin layer
to change Unix domain credentials
based on grid identity and attribute information

you can think of it as
> ‘a replacement for the gatekeeper’
> ‘a griddy version of Apache’s suexec’
> ‘a program wrapper around LCAS, LCMaps or GUMS’
What gLExec does …

- User grid credential (subject name, VOMS, ...)
- Command to execute
- Current uid allowed to execute gLExec

Authorization (‘LCAS’) check white/blacklist VOMS-based ACLs is executable allowed?

Credential Acquisition voms-poolaccount localaccount GUMS, ...

LCMAPS

‘do it’ LDAP account posixAccount AFS, ...

Execute command with arguments as user (uid, pgid, sgids ... )
V Pieces of the solution

VO supplied pilot jobs must observe and honour
the same policies the site uses for normal job execution
(e.g. banned individual users)

Three pieces that go together:
> glexec on the worker-node deployment
  > the mechanism for pilot job
to submit themselves and their payload to site policy control
  > give ‘incontrovertible’ evidence of who is running on which node at any one
time (in mapping mode)
    • gives ability to identify individual for actions
    • by asking the VO to present the associated delegation for each user
> VO should want this
  • to keep user jobs from interfering with each other, or the pilot
  • honouring site ban lists for individuals may help in not banning the entire VO in
case of an incident
V  Pieces of the solution

> glexec on the worker-node deployment

> keep the pilot jobs to their word
  > mainly: monitor for compromised pilot submitters credentials
  > process or system call level auditing of the pilot jobs
  > logging and log analysis

> gLExec cannot to better than what the OS/batch system does
  > ‘internal accounting should now be done by the VO’
    • the regular site accounting mechanisms are via the batch system, and these will see the pilot job identity
    • the site can easily show from those logs the usage by the pilot job
    • accounting based glexec jobs requires a large and unknown effort
  > time accrual and process tree remain intact across the invocation
    • but, just like today, users can escape from both anyway!
V But all pieces should go together

1. *glexec on the worker-node deployment*

2. way to keep the pilot jobs submitters to their word
   > mainly: monitor for compromised pilot submitters credentials
   > system-level auditing of the pilot jobs,
     but auditing data on the WN is useful for incident investigations only

3. ‘internal accounting should be done by the VO’
   > the regular site accounting mechanisms are via the batch system, and
     these will see the pilot job identity
   > the site can easily show from those logs the usage by the pilot job
   > making a site do accounting based glexec jobs is non-standard,
     and requires non-trivial effort
gLExec deployment modes

- Identity Mapping Mode – ‘just like on the CE’
  - have the VO query (and by policy honour) all site policies
  - actually change uid based on the true user’s grid identity
  - enforce per-user isolation and auditing using uids and gids
  - requires gLExec to have setuid capability

- Non-Privileged Mode – declare only
  - have the VO query (and by policy honour) all site policies
  - do not actually change uid: no isolation or auditing per user
  - Pilot and framework remain vulnerable
  - the gLExec invocation will be logged, with the user identity
  - does not require setuid powers – job keeps running in pilot space

- ‘Empty Shell’ – do nothing but execute the command...
V Installation

- Actually only identity mapping mode really helps

  Otherwise

   - back-compromise (and worm infections) remain possible
   - attributing actions to users on WN is impossible (that needs a uid change)
Centralizing Authorization in the site
Available middleware: GUMS and SAZ, Argus, SCAS
Interoperability through common protocols

TOWARDS CENTRAL CONTROL
What Happens to Access Control?

So, as the workload binding get pushed deeper into the site, access control by the site has to become layered as well ...

... how does that affect site access control software and its deployment?
Site Access Control today

PRO already deployed
no need for external components, amenable to MPI

CON when used for MU pilot jobs, all jobs run with a single identity
end-user payload can back-compromise pilots, and cross-infect other jobs
incidents impact large community (everyone utilizing the MUPJ framework)
V Centralizing decentralized SAC

Aim: support consistently
> policy management across services
> quick banning of bad users
> coordinated common user mappings (if not WN-local)

Different options to implement it ...
Central SAC management options

> Regular site management tools (CFengine, Quattor, etc)
  > Addresses site-wide banning in a trivial and quick way
  > Does not address coordination of mapping (except NFS for the gridmapdir)

> GUMS (use the new interoperability version 2)
  > database with users available at all times, but it is not ‘real-time’
  > Extremely well stress tested

> Argus (use at least v1.1 or above)
  > Supports all common use cases, with resilience in mind
  > in addition also grid-wide policy distribution and banning!

> SCAS (transitional)
  > service implementation of the LCAS/LCMAPS system
  > Client can talk natively also to GUMS v2 and GT

> All together can be used in composition to support more use cases
  > e.g. add support for AFS token acquisition via LCMAPS, plain-text ban-lists
  > shared with storage via LCAS, grid-wide banning via Argus, joint GACL
  > support with the current WMS, …
Centralizing access control in M/W

**PRO**
- Single unique account mapping per user across whole farm, CE, and SE*
- Can do instant banning and access control in a single place

**CON**
- Need remedy single point of failure (more boxes, failover, i.e. standard stuff)

*Credential validation is still done on the end-nodes for protocol reasons*

---

* of course, central policy and distributed per-WN mapping also possible!
Talking to an AuthZ Service: standards

- Existing standards:
  - **XACML** defines the XML-structures that are exchanged with the PDP to communicate the security context and the rendered authorization decision.
  - **SAML** defines the on-the-wire messages that envelope XACML's PDP conversation.

- The Authorization Interoperability profile augments those standards:
  - Standardize names, values and semantics for common-obligations and core-attributes such that our applications, PDP-implementations and policy do interoperate.
V Two Elements for interop

> Common *communications* profile
  > Agreed on use of SAML2-XACML2

> Common *attributes and obligations* profile
  > List and semantics of attributes sent and obligations received between a ‘PEP’ and ‘PDP’
  > Now at version 1.1
Aims of the authz-interop project

> Provide interoperability within the authorization infrastructures of OSG, EGEE, Globus and Condor

> See www.authz-interop.org

Through

> Common communication protocol

> Common attribute and obligation definition

> Common semantics and actual interoperation of production system

So that services can use either framework and be used in both infrastructures
An XACML AuthZ Interop Profile

An XACML Attribute and Obligation Profile for Authorization Interoperability in Grids

OGS: Mine Altmay, Keith Chadwick, Gabriele Carzoglia (editor), Ted Hesselroth, John Hover,
Tanya Levashina, Jay Packard, Valerie Segree, Igor Sfiligoi, John Weigand
EGEE: Yuri Demchenko, Oscar Koerw, Chad La Jole, Hakon Sagelang
EGEE / INFN: Vincenzo Ciaschini, Andrea Ferraro, Alberto Forti, Valerto Venturi
Globus: Rachana Ananthakrishnan, Joe Bester, Frank Siebenlist
Condor: Ian Alderman, Zach Miller

Opt 09, 2008
Document Version: v1.1

Change log
v1.0: First release of the standard. May 16, 2008

Introduction
Goal of the Authorization Interoperability activity is providing interoperability between middleware and authorization infrastructures. This is achieved by designing and implementing an authorization protocol common to OSG VO services, EGEE, Globus, and Condor. This protocol is based on the SAML profile of XACML v2.0 [XACML]. The C library that implements the profile is provided by the Globus Toolkit security group; the JAVA library by the SWITCH group of EGEE.

The authorization protocol is used by Policy Enforcement Points (PEP), i.e. resource gateways, to interact with Policy Decision Points (PDP), i.e. repository of authorization policies. For each access request, the PDP informs the PEP on whether access is granted or denied and the conditions to be enforced if access if granted. These conditions are expressed in the form of XACML Obligations and are the mechanism to restrict privileges at Grid resources.

> Authorization Interoperability Profile based on the SAML v2 profile of XACML v2

> Result of a 1yr collaboration between OSG, EGEE, Globus, and Condor

> Releases:
v1.1 → 10/09/08
v1.0 → 05/16/08
Most Common Obligation Attributes

> **UIDGID**
  - **UID** (integer): Unix User ID local to the PEP
  - **GID** (integer): Unix Group ID local to the PEP

> **Secondary GIDs**
  - **GID** (integer): Unix Group ID local to the PEP (Multi recurrence)

> **Username**
  - **Username** (string): Unix username or account name local to the PEP.

> **Path restriction**
  - **RootPath** (string): a sub-tree of the FS at the PEP
  - **HomePath** (string): path to user home area (relative to RootPath)

> **Storage Priority**
  - **Priority** (integer): priority to access storage resources.

> **Access permissions**
  - **Access-Permissions** (string): “read-only”, “read-write”

*see document for all attributes and obligations*
What has been achieved now

> All profiles written and implemented
> Common libraries available in Java and C implementing the communications protocol
> Common handlers for Joint Interoperable Attribute and Obligations
> Integrated in all relevant middleware in EGEE and OSG:
  > Clients: lcg-CE (via LCMAPS scasclient), CREAM and gLExec (ditto), GT pre-WS gram (both prima and LCMAPS), GT GridFTP, GT4.2 WS-GRAM, dCache/SRM
  > Servers: GUMS, SCAS, Argus (variant protocol)
> Other (lower-prio) components in progress
  > SAZ, RFT, GT WS native-AuthZ, Condor (& -G), BeStMan
Application links LCMAPS dynamically or statically, or includes Prima client.
Local side talks to SCAS using a variant-SAML2XACML2 protocol:
- with agreed attribute names and obligation between EGEE/OSG
- remote service does acquisition and mappings
- both local, VOMS FAQN to uid and gids, etc.
Local LCMAPS (or application like gLExec) does the enforcement.
V Talking to SCAS

> From the CE
  > Connect to the SCAS using the CE host credential
  > Provide the attributes & credentials of the service requestor, the action
    (“submit job”) and target resource (CE) to SCAS
  > Using common (EGEE+OSG+GT) attributes
  > Get back: yes/no decision and uid/gid/sgid obligations

> From the WN with gLEexec
  > Connect to SCAS using the credentials
    of the pilot job submitter
    An extra control to verify the invoker of gLEexec
    is indeed an authorized pilot runner
  > Provide the attributes & credentials of the service requestor, the action
    (“run job now”) and target resource (CE) to SCAS
  > Get back: yes/no decision and uid/gid/sgid obligations

> The obligations are now coordinated between CE and WNs
SCAS Supported services & protocols

- SCAS communicates based on a few standards and the joint “Authorization Interoperability” profile
  - Supported by Globus, EGEE/gLite 3.x, VO Services/OSG, dCache
  - Defined also common wire protocol
  - Common naming of obligations such as uid/gid, rootPath

- Compatible software
  - Globus gatekeepers, lcg-CE
  - gLExec (on WNs and on CREAM-CEs)
  - dCache > 1.9.2-4
  - GT GridFTP
  - GT4.2 WS-GRAM, GRAM5 (to be tested)
V GUMS and SAZ

Legend
- AuthZ Components
- Not Officially In OSG
- VO Management Services

graphic: Dave Dykstra, Fermi National Accelerator Laboratory, CHEP, March 2009
V Interoperability achievements

graphic: Gabriele Garzoglio, FNAL
V Argus service

![Diagram of Argus service](image)

**graphic:** MJRA1.4 (EGEE-II) gLite security architecture, Oct 2008, Christoph Witzig
V  Argus services and daemons

> Administration Point
Formulating rules through CLI and/or file-based input

> Decision Point
Evaluating a request from a client based on the rules

> Enforcement Point
Thin client part and server part: all complexity in server part

> Runtime Execution Environment
Under which env. must I run? (Unix UID, GID, ...)

Graphic: Christoph Witzig, SWITCH and EGEE
V Capabilities

> Enables/eases various authorization tasks:
  > Banning of users (VO, WMS, site, or grid wide)

> Composition of policies – e.g.
  CERN policy + experiment policy + CE policy
  + OCST policy + NGI policy=> Effective policy

> Support for authorization based on more detailed information about the job, action, and execution environment
  > Support for authorization based on attributes other than FQAN
  > Support for multiple credential formats (not just X.509)
  > Support for multiple types of execution environments
  > Virtual machines, workspaces, ...

https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework
**V** Introduction of the service in gLite

> Focus is on computing services (again ...)
  > Initial introduction through gLExec on the WN
  > As a new LCMAPS plug-in
    (used in conjunction with the others, esp. verify-proxy)
  > With OSCT ban list

> standards expressibility
  > ‘PIP, PEP, PAP, PDP’, and SAML
  > XACML policies and attributes
  > But with a simplified language 😊

> v1.1 released in Feb 2010
  > Contains important fixes – use at least this one or better

---

**Graphic:** Christoph Witzig, SWITCH and EGEE
gLEexec with Argus

> ‘just another call-out from LCMAPS’

```
# LCMAPS config file for glexec generated by YAIM

# Plugin definitions:
posix_enf = "lcmaps_posix_enf.mod"
    " -maxuid 1" " -maxpgid 1" " -maxsgid 32"
verifyproxy = "lcmaps_verify_proxy.mod"
    " -certdir /etc/grid-security/certificates"

pepc = "lcmaps_c_pep.mod"
    "--pep-daemon-endpoint-url https://mient.nikhef.nl:8154/authz"
    "--resourcetype wn"
    "--actiontype execute-now"
    "--capath /etc/grid-security/certificates"
    "--pep-certificate-mode implicit"

# LCMAPS Execution Policies:
argus:
verifyproxy -> pepc
pepc -> posix_enf
```

/opt/glite/etc/lcmaps/lcmaps-argus.db
Argus Supported services & protocols

- Argus communicates based on many of the well-known standard protocols
  - Same common wire communications protocol as Globus, EGEE/gLite 3.x, VO Services/OSG, and SCAS
  - Naming derived from but slightly different from the Joint Profile
    but will not yet work with AuthZ Interop attribute profile compliant apps

- Compatible software
  - gLEexec (on WNs and on CREAM-CEs)
  - All LCMAPS capable services via common PEP-C plugin
  - GT4 pre-WS gatekeeper via dedicated GT4 authZ call-out
  - Scale-out to WMS and storage services foreseen
Combining services

> If you want, e.g.,
banning from Argus and mapping done locally?
> Configure Argus service to not run a pool-account map
> Then, chain a lcmaps_c_pep plugin and a voms-poolaccount in sequence, followed by posix_enf

```sh
# Policies
good_account_mapping:

verifyproxy -> pepc
pepc -> vomslocalgroup
vomslocalgroup -> vomslocalaccount | localaccount
vomslocalaccount -> posix_enf | vomspoolaccount
vomspoolaccount -> posix_enf
localaccount -> posix_enf | poolaccount
poolaccount -> posix_enf
```
MyProxy
Renewal daemons
What About VOMS

LONG RUNNING JOBS
MyProxy in EGEE

> EGEE security based on proxy certificates
  > often carrying VOMS attribute certificates

> MyProxy used for several purposes:
  > Solution for portals (P-GRADE, Genius)
    • a common way of using MyProxy
  > Long-running jobs and data transfers
    • credential renewal

http://myproxy.ncsa.uiuc.edu/

Slides based on: Ludek Matyska and Daniel Kouril, CESNET
Long-running Jobs

> Jobs require valid credentials
  > e.g. to access GridFTP data repositories on the user’s behalf
  > these operations must be secured, using the users‘ credentials

> Job's lifetime can easily exceed the lifetime of a proxy
  > consider waiting in the queues, possible resubmissions, computation time, data transfers, etc.
  > also VOMS certificates have limited lifetime

> Impossible to submit a job with sufficiently long credentials
  > the overall job lifetime not known in advance
  > violation of the meaning of short-time proxies
  > increased risk when the credential is stolen
  > might be unacceptable for the end resources

> How to provide jobs with a valid short-lived credential throughout their run?
Proxy Renewal Service

PUT proxy

submit

MyProxy Service

GET proxy

VOMS

submit, renew

Renewal Service

Resource Management

Fabric Management

Job

Gatekeeper

Slides based on: Ludek Matyska and Daniel Kouril, CESNET
Proxy Renewal Service

- Ensures that jobs always have a valid short-time proxy
- Users have full control over their proxies and renewal
  - Using the MyProxy repository
- Support for VOMS
- All operations are logged
  - allows an audit
- Stolen credentials can't be renewed easily
  - the WMS credential are necessary for renewal
- An older (still valid) proxy must be available for renewal
  - reduces the risk when services are compromised
- Developed in EU Datagrid, in production use in EGEE

Slides based on: Ludek Matyska and Daniel Kouril, CESNET
MyProxy and Trust Establishment

- Relationship between MyProxy and its client is crucial
  - clients must be authorized to access the repository

- So far trust based on a static configuration
  - each service and client must be listed
  - regular expressions aren’t sufficient
  - a subject name of a service must be added on each change or addition

- VOMS support introduced recently
  - generated by needs of EGEE
  - allows to specify VOMS attributes (roles, groups) instead of specifying identity
  - requires adding service certificates to VOMS machinery
Access Control semantics
Breakdown of the container model
Legacy forever: mapping grid storage onto Unix semantics
The DPM model

DATA STORAGE
Storage: Access Control Lists

> Catalogue level
  > protects access to meta-data
  > is only advisory for actual file access
    unless the storage system only accepts connections from a trusted agent that does
    itself do a catalogue lookup

> SE level
  > either natively (i.e. supported by both the SRM and transfer services)

> SRM/transfer level
  > SRM and GridFTp server need to lookup in local ACL store for each transfer
  > need “all files owned by SRM” unless underlying FS supports ACLs

> OS level?
  > native POSIX-ACL support in OS would be needed
  > Mapping would still be requires (as for job execution)
Grid ACL considerations

> Semantics
  > Posix semantics require that you traverse *up* the tree to find all constraints
  > behaviour both costly and possibly undefined in a distributed context
  > VMS and NTFS container semantics are self-contained
  > taken as a basis for the ACL semantics in many grid services

> ACL syntax & local semantics typically Posix-style
‘Container abstraction’ breakdown

Policy Database

- PIDA: group 1: read; group 2: all; group 3: none; user7: read
- PODB: group 1: read, write; group 2: all; group 3: all

Policy Engine

- (5) Pass policy ID, subject, object, action
- (6) Query policies for PIDs
- (7) Permit or deny

LRC

- LFN1
- PIDA
- LFN2
- PODB

(3) Request PIDs for logical names
(4) PIDs

Custom PDP

(2) Custom auth callout (includes client request)
(8) Permit or deny

GT4 Authorization Framework

(1) Client Request

(9) If permitted, pass client request to LRC

database consistency

graphic: Ann Chervenak, ISI/USC, from presentation to the Design Team, Argonne, 2005
Embedded access control: dCache

- voms-proxy-init
  Proxy with VO
  Membership | Role attributes

- SAML2XACML2 interop protocol
  GUMS, SCAS, &c

Graphic: Frank Wurthwein, CHEP2006 Mumbai
Legacy persists, though

> dCache/gPlazma maps back to
  > Unix username
  > ‘root path’

> Files stored with *Unix* uid and gid
  > Can have local access!
  > But doing VOMS-based ACLs over simple Unix ACLs results in a combinatorial group explosion

Graphic: Frank Wurthwein, CHEP2006 Mumbai
Grid storage access control

- Use ‘grid’ identity and attributes to define ACLs

- With ‘POSIX’ semantics
  - So traversal based, not object based
  - Needs ‘good’ database schema to store ACLs&metadata

- Example: DPM “Disk Pool Manager”
  - See
    https://twiki.cern.ch/twiki/bin/view/EGEE/GliteDPM
> All disk-based (data) files owned by a generic ‘dpm’ user
> Meta-data, locations, ownership, ACLs: all in a database

graphics: ‘ACLs in Light Weight Disk Pool Manager’ MWG 2006, Jean Philippe Baud, CERN
Virtual Ids and VOMS integration

- DNs are mapped to virtual UIDs: the virtual uid is created on the fly the first time the system receives a request for this DN (no pool account)
- VOMS roles are mapped to virtual GIDs
- A given user may have one DN and several roles, so a given user may be mapped to one UID and several GIDs
- Currently only the primary role is used in LFC/DPM
- Support for normal proxies and VOMS proxies
- Administrative tools available to update the DB mapping table:
  - To create VO groups in advance
  - To keep same uid when DN changes
  - To get same uid for a DN and a Kerberos principal
V DPNS mapping tables

CREATE TABLE Cns_groupinfo (  
gid NUMBER(10),  
groupname VARCHAR2(255));

CREATE TABLE Cns_userinfo (  
userid NUMBER(10),  
username VARCHAR2(255));

> included in GridFTP through ‘dli’ plugin mechanism,  
and in SRM through call-outs to dpns

Slides and graphics: ‘ACLs in Light Weight Disk Pool Manager’ MWSG 2006, Jean Philippe Baud, CERN
V Access Control Lists

> LFC and DPM support Posix ACLs based on Virtual Ids
  > Access Control Lists on files and directories
  > Default Access Control Lists on directories: they are inherited by the sub-directories and files under the directory

> Example
  > dpns-mkdir /dpm/cern.ch/home/dteam/jpb
  > dpns-setacl -m d:u::7,d:g::7,d:o:5 /dpm/cern.ch/home/dteam/jpb
  > dpns-getacl /dpm/cern.ch/home/dteam/jpb

# file: /dpm/cern.ch/home/dteam/jpb
# owner: /C=CH/O=CERN/OU=GRID/CN=Jean-Philippe Baud 7183
# group: dteam
user::rwx
group::r-x              #effective:r-x
other::r-x
default:user::rwx
default:group::rwx
default:other::r-x
SPECIALISED MIDDLEWARE

Hydra distributed key store

SSSSS
V Encrypted Data Storage

Medical community as the principal user
- large amount of images
- privacy concerns vs. processing needs
- ease of use (image production and application)

Strong security requirements
- anonymity (patient data is separate)
- fine grained access control (only selected individuals)
- privacy (even storage administrator cannot read)

Described components are under development

Slides based on Akos Frohner, EGEE and CERN
V Accessing medical images

- image ID is located by AMGA
- key is retrieved from the Hydra key servers (implicitly)
- file is accessed by SRM (access control in DPM)
- data is read and decrypted block-by-block in memory only (GFAL and hydra-cli)--- useful for all

Still to be solved:
- ACL synchronization among SEs

Slides based on Akos Frohner, EGEE and CERN
V Exporting Images

“wrapping” DICOM:

- anonymity: patient data is separated and stored in AMGA
- access control: ACL information on individual files in SE (DPM)
- privacy: per-file keys
  - distributed among several Hydra key servers
  - fine grained access control

Image is retrieved from DICOM and processed to be “exported” to the grid.
Hydra key store theory, and SSSS

Keys are split for security and reliability reasons using Shamir's Secret Sharing Scheme (org.glite.security.ssss)

- standalone library and CLI
- modified Hydra service and Hydra client library/CLI
- the client contacts all services for key registration, retrieval and to change permissions
  - there is no synchronization or transaction coordinator service

$ glite-ssss-split-passwd -q 5 3 secret
137c9547aba101ef 6ee7adbbaacac1ef 1256bcc160eda592
    fdabc259cdfbacc9 3113be83f203d794
$ glite-ssss-join-passwd -q 137c9547aba101ef NULL \n1256bcc160eda592 NULL 3113be83f203d794
  secret
Example: integration into DPM

- `lcg-cp -bD srmv2 srm://dpm.example.org:8446/srm/managerv2?`
  - SFN=/dpm/example.org/home/biomed/mdm/<ID> file:picture.enc
- `glite-eds-decrypt <ID> picture.enc picture`
- `glite-eds-get -i <ID> rfio://dpm/example.org/home/biomed/mdm/<ID> picture`
  - file is opened via gfal_open()
  - decryption key is fetched for <ID>
  - loop on gfal_read(), glite_eds_decrypt_block(), write()

'glite-eds-get' is a simple utility over the EDS library.
Summary and last words

FROM HERE?
V Summary

> Security middleware is everywhere
  > An integral part of almost any grid service
  > And (but ...) implemented in a myriad of ways

> Most of the core capabilities are there
  > VOMS based access, banning on VO or DN
  > But methodology varies,
    and the documentation is not well read or disseminated

> New frameworks will help you manage security at the site
  > Deal with new usage patterns and novel risk surfaces
  > We’re getting there with interop
  > No reason to wait: regular site management can already help a lot

> And (or: but): we’re far from done ...
QUESTIONS?