Enabling Grids for E-sciencE

L&B and JP authorization

Aleš Křenek, on behalf of CESNET JRA1 team

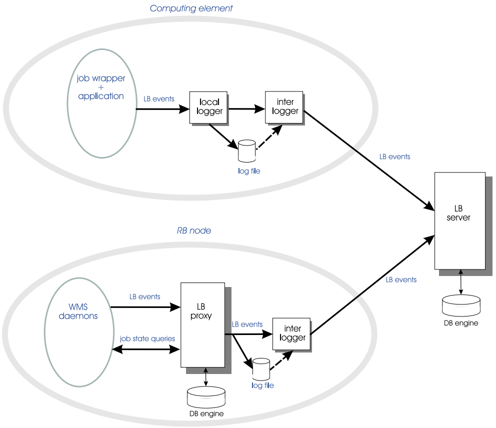
www.eu-egee.org

Information Society

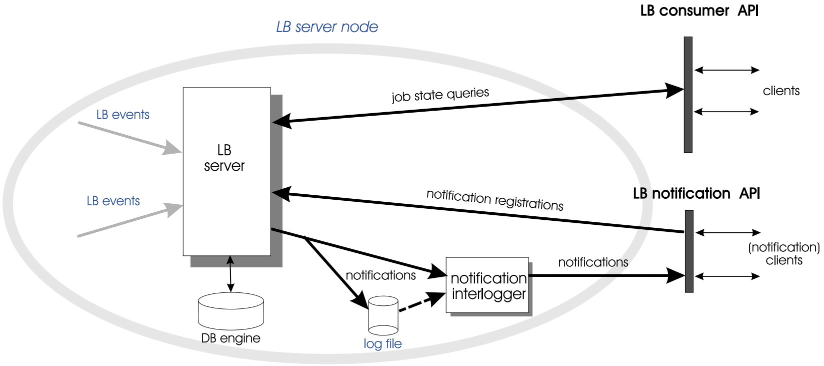
Overview

- What we have today
  - services overview
  - mutual components and user interactions
  - authorization done
- What should be done
- How shall we do it
- What we expect from authz service

L&B overview


Computing element
job wrapper
+
application
LB events
local logger
inter logger
log file
LB events
LB server
RB node
WMS daemons
LB events
job state queries
LB proxy
LB events
inter logger
log file
DB engine
LB events
DB engine

L&B overview (2)



Major usecases

- job registration
  - determines who is allowed to use particular L&B server
- storing events asynchronously
  - store-and-forward protocol
  - user credentials available for 1st hop only
- sharing jobs among users
  - both read and write access (annotations)
  - job, user, and VO levels
  - privileged access (VO admin)
  - restricted views (site admin sees jobs executed at her site)
- user identity change
  - pilot jobs + glexec

Abstract authz library

- used to describe requirements in this presentation
- `magic_authz(usercred,op,{jobspec},{obligations})`
  - usercred: how the user authenticated (DN, VOMS groups/roles, . . .)
    - as little Globus dependence as possible
    - cannot rely on delegated credentials
  - op: currently REGISTER, STORE, and QUERY
  - jobspec: name=value pairs describing the job; jobid, owner, VO, . . .
    - must be extensible
  - obligations: finer grain control
- plugin framework
  - plugins called one after another
  - dynamically configurable
  - not all requirements are known now

Abstract authz library (2)

- 3 return values
  - ALLOW, DENY, DONTKNOW
  - useful for configuring plugins
  - can make sense for overall return value of `magic_authz()`
- target performance
  - 1M jobs/day $\sim$ 14 jobs/s $\sim$ 100–500 events/s
- all decisions must be done locally
- supported by authz infra
  - eg. policy exchange and pre-evaluation
- calling even site-central authz service synchronously is unfeasible

User access

- authorization granularity
  - server policies
  - per user
  - per job
  - finer control (events, their fields) enforced via obligations
- `magic_authz()` invocation
  - in query processing, for each matching job
  - the more is cached inside, the better

User access (2)

- owner always allowed
  - jobspec fields: jobOwner
  - evaluation: comparison `jobspec{jobOwner} == usercred{DN}`

User access (2)

- owner always allowed
  - jobspec fields: jobOwner
  - evaluation: comparison `jobspec{jobOwner} == usercred{DN}`
- specific ACL (GACL or XACML) maintained with job
  - jobspec fields: jobACL
  - evaluation: parse jobACL, check wrt. usercred
  - favour VOMS, SAML . . . in usercred

User access (3)

- user-level ACL
  - eg. Alice allows Bob to query her jobs
  - jobspec fields: userACL
  - evaluation: same as per-job ACL
  - needs maintenance of userACL in L&B– TBD later if required

User access (3)

- user-level ACL
  - eg. Alice allows Bob to query her jobs
  - jobspec fields: userACL
  - evaluation: same as per-job ACL
  - needs maintenance of userACL in L&B– TBD later if required
- L&B server policies
  - eg. all jobs of some VO readable by 'Admin' VOMS role of the VO
  - eg. job executed at a site readable by the site admin
  - specified in L&B server config
  - thinkable jobspec fields: VO, destination (site), . . .

User access (4)

- simple notifications
  - jobid(s) known on subscription
  - same authz evaluation as for queries
  - evaluated on subscription/refresh
  - only DN checked on delivery
  - revocation effective after subscription expires
- complex notifications
  - authz can't be completely evaluated on subscription
  - postponed to delivery
  - details not clear yet

Adding new policies

- can be expressed in existing mechanism
  - configuration change only
- or one or more of the following may be required
  - provide new authz plugin
  - extend set of passed jobspec fields
  - implement specific obligations

Asynchronous event delivery

- WM – L&B proxy
  - local trust, no authz
- job wrapper – logd
  - not enough job information for full authz
  - authentication only here, record user's identity
- interlogd – L&B server
  - `magic_authz()` called on event arrival
  - both interlogd and orignal sender identity must be checked
  - optimistic strategy
    - retrieve job state information
    - extract jobspec fields and call `magic_authz(...,STORE,...)`
    - calling back for job state is less efficient
    - retrieved job state is used if authz passed

Asynchronous event delivery (2)

- from WMS
  - list of trusted WMS's at L&B
  - VOMS-based – needs host credentials in VOMS
  - complex rules possible (eg. this WMS for this VO only)
- from sites
  - "trusted service" approach difficult to manage
  - messages (events) signed with user credentials may be appropriate

Job registration

- synchronous operation, L&B library – server directly
- authenticated with user credentials
- `magic_authz(...,REGISTER,...)` called on server
- specific policies may be applied
  - eg. allow users of particular VO only

Events signing

- performance problem in general
- simplifies most service-service authz problems
- per-job symmetric key
  - generated on registration, signed with user credentials
  - used mainly on WMS
  - eventually propagated with the job and used to sign further events
- used to secure event delivery only
  - signatures checked on L&B server on event arrival
  - not stored for further checks (authoritative timestamps would be required)

Available technology

- Gridsite/GACL and `security.acl-parser`
  - both express and evaluate ACL's
  - not clear what is the prefered way
- LCAS
  - simple API, well tested framework
  - good starting point, some extensions needed
- VOMS
  - suitable for local authz evaluation
- gPbox
  - promissing for complex policy specification, policy combinations
- gJAF
  - Java world, no foreseen easy integration

Job Provenance

- similar approach to L&B
  - call `magic_authz()` whenever appropriate
- Primary storage (JPPS)
  - mostly privileged access and trusted services
  - long-lived data – problem of users changing identities
- ftp interface of JPPS
  - bound to “open URL” operation of JPPS WS interface
  - only DN checked
- JP index servers
  - trusted
    - get any data from JPPS
    - per-job authorization at JPIS
  - run by user
    - per-job authorization at JPPS

Requirements summary

- `magic_authz` implementation
  - plugin framework for configurability and future extensions
  - fast local evaluation
- specification of plugin interface
  - we will have to provide our plugins
- service-service authz
  - needs VOMS attributes for service credentials