Experience with gLEexec working with SCAS and Argus

To cite this article: Angela Poschlad 2011 J. Phys.: Conf. Ser. 331 062042

View the article online for updates and enhancements.

Related content
- glideinWMS experience with plexec
  I Sfiligoi, D C Bradley, Z Miller et al.
- Performance of combined production and analysis WMS in DIRAC
  Stuart Paterson, Joel Closier and the Lhcb Dirac Team
- arcControlTower: the System for Atlas Production and Analysis on ARC
  Andrej Filipi and the Atlas collaboration
Experience with gLExec working with SCAS and Argus

Angela Poschlad
Karlsruhe Institute of Technology (KIT), Steinbuch Centre for Computing (SCC)
E-mail: angela.poschlad@kit.edu

Abstract. With the concepts of grid pilot jobs new submission frameworks were built by various communities. To ensure security requirements gLExec can be used operating in setuid mode. The two available backends for gLExec - SCAS and Argus - provide unified user mapping and offer an easy user handling. In this contribution the experiences at GridKa, the German WLCG Tier-1 Center, with SCAS and Argus, differing service features and their pros and cons are discussed. Both services have been tested for their performance and reliability as well as the integration in the existing site concept.

1. Introduction
The job submission can take place via different tools and protocols but in the end the grid job arrives at the site on a Worker Node (WN) where it is executed. The normal user job workflow is depicted on the left of Figure 1. The submitting user is mapped according to his grid proxy to the dedicated user pool account. If he/she has not yet got one assigned, the user will get a free account. As soon as the job arrives on a WN, the users code is executed. In this scenario the site has the ability to set fairshares to favour or disadvantage someone or some group. Normally these fairshares are determined between the communities organized in Virtual Organizations (VOs) and the sites. In some regions national users get an extra share if they use resources funded by national agencies.

Figure 1. Illustrated workflow of grid jobs: On the left, a normal user grid job arrives on the WN. In the middle, the workflow shows a pilot grid job pulling the user job from the pilot framework. On the right, the pilot job additionally calls SCAS/Argus via gLExec to map the user to a different account.
Pilot jobs on the contrary do not contain users code but the pilot codes to get the user jobs (payload jobs) from a separate pilot framework. When a pilot job arrives on a WN, the pilot user is mapped to a pilot pool account dedicated to the pilot’s proxy. After starting, the pilot job checks the environment and asks the job submission pilot framework whether an adequate payload job is waiting as illustrated in the middle of Figure 1. If there is no job it will exit. If the pilot job finds a fitting payload, it will be executed under the pilot pool account. Pilot jobs allow the VO to prioritise its grid jobs on their own and skip the site mechanisms.

To avoid grid jobs other than the pilot job to be executed under the pilot account, the new payload has to be mapped to a new pool account. This can be done using gLExec (see Figure 1 right). For executing setuid, root privileges are necessary. gLExec is granted certain root privileges and therefore its usage is highly restricted to pilot accounts. Using gLExec will result in the payload being executed under a different user ID given by SCAS, Argus or a local authorization. After the user ID change the virtual grid job is executed.

In section 2 the motivation for using gLExec and a brief description of the functionality is given followed by a short overview of the integration in GridKa’s site concept. Section 4 contains the results of different performance tests and the advantages and disadvantages of the tools are discussed in section 5.

2. Why support gLExec?

Why should a site use software with root privileges? For this new security issue some old issues can be avoided.

2.1. Motivation

With VOs using Single User Pilot Jobs (SUPJs) or Multi User Pilot Jobs (MUPJs) such as PanDA for ATLAS [2] or Alien for ALICE [3], sites lose their ability to trace user activity to a specific user. gLExec allows the sites to log the Distinguished Name (DN) of the real user. This enables the sites to include all users in the site concepts, e.g. security, logging. The VO benefits through protection of the user and pilot proxies by using separated accounts. During the European project ‘Enabling Grid for E-Science (EGEE)’ the Joint Security Policy Group released a policy for MUPJs [4].

These job submission frameworks hold risks. The execution of a payload with the pilot proxy implies that this proxy is readable by the payload. A malicious payload may copy the pilot proxy and reuse it in an improper way. If the pilot proxy is deleted before starting a payload it seems to be safe at first sight, but as described above, the pilot user is always mapped to the same local account on the WN. So it can happen that two jobs from different users are executed by two pilot jobs but on the same WN. When using a shared home directory this can happen even if the jobs are executed on different WNs. This can be avoided by using a local file space such as /tmp. Therefore Single User Pilot Jobs hold the same risk as Multi User Pilot Jobs. In the latter case a payload may leave an orphan behind which can act on the following payloads or read the next user proxy and steal it. These threats can be met by using gLExec.

2.2. How does gLExec work?

gLExec can either operate in setuid mode or just logging the user DN. The mapping of the setuid mode requires a central authorization tool or a local mapping can be made [8]. The client includes a failover mechanism. The two available backends for gLExec - SCAS and Argus - provide unified user mapping and offer an easy user handling. After successful authorization the backends deliver a Unix user ID and group ID.
2.3. How does SCAS work?
SCAS stands for Site Central Authorization Service [7]. The service decides for each mapping request according to the rules for the local credential mapping service (LCMAPS) and the local centre authorization service (LCAS), the current authorizing procedure for different gLite middleware nodes, i.e. there is one policy for all calls.

2.4. How does Argus work?
Argus (name comes from the Greek mythology) offers semantic-aware mapping [1], i.e. the mapping depends on who wants to be mapped from where. For Argus policies have to be defined before any mapping is allowed. The request handling is done via several steps:

- PEP: Policy Enforcement Point,
- PDP: Policy Decision Point,
- PAP: Policy Administration Point.

In addition Argus offers further features. The most beneficial functionality is the chance of importing user ban lists from central and/or regional instances. Another feature is the decision memory which helps to improve the performance. On the web the results of a performance test for the load and lifetime of the service is available for Argus [6]. The author is not aware of any documented comparative test for SCAS and Argus systems.

3. Including SCAS and Argus in GridKa’s site concept
The central administration tool in GridKa is cfengine. Important files such as CA certificates are distributed via a hierarchical cfengine server structure. The user banning is also maintained with the help of this tool. All banned users are deleted from grid-mapfiles and added to the ban lists for the different services. For SCAS, the same file as for the file transfer via gridftp is used. Argus has another banning mechanism which is not as easy to implement. Everytime the list of banned users has to be updated all policies have to be removed first. This results in a short unavailability in the service. Therefore Argus was not included in the central procedure but it is maintained by hand.

4. Performance tests
The tests ran on the GridKa cluster with the gLExec client version glite-security-glexec-0.7.0-1.sl5. For SCAS and Argus equivalent Virtual Machines (VM) have been used (Table 1).

| OS          | Scientific Linux 5.3 (Boron) |
|-------------|-----------------------------|
| Memory      | 1024 MB                     |
| CPU         | 1 core (2.6 GHz)            |
| Argus       | glite-ARGUS-1.1.0-1.sl5     |
| SCAS        | glite-SCAS-3.2.2.0          |
| gLExec      | glite-security-glexec-0.7.0-1.sl5 |

Table 1. Specification of the used hardware and involved middleware versions.

4.1. Performance with rising request frequency
For this test, the request frequency was varied between 0.1 Hz and 200 Hz followed by undelayed requests (referred to as an ‘unlimited’ request rate) as listed in column “variation 1” of Table 2. The input frequencies are shown in Figure 2 as the green line together with the achieved response frequencies for SCAS represented by the red line and Argus indicated by the blue line.
Table 2. Performance tests: The middle column gives the different input frequencies for the authorisation test. In the right column the variated usage of proxies is noted.

| Test interval | variation 1 | variation 2 |
|---------------|------------|-------------|
| 0 - 15 min    | 0.1 Hz     | 2 valid proxies at 3 Hz |
| 15 - 30 min   | 0.3 Hz     | 1 valid and 1 invalid proxy at 3 Hz |
| 30 - 45 min   | 0.5 Hz     | 3 valid proxies at 3 Hz |
| 45 - 60 min   | 1 Hz       | 2 valid and 1 invalid proxy at 3 Hz |
| 60 - 75 min   | 3 Hz       | 4 valid proxies at 3 Hz |
| 75 - 90 min   | 5 Hz       | 3 valid and 1 invalid proxies at 3 Hz |
| 90 - 105 min  | 7 Hz       | 4 valid and 1 invalid proxies at 3 Hz |
| 105 - 120 min | 10 Hz      | 4 valid and 1 invalid proxies at 5 Hz |
| 120 - 135 min | 40 Hz      | 4 valid and 1 invalid proxies at 7 Hz |
| 135 - 150 min | 65 Hz      | 4 valid and 1 invalid proxies at 10 Hz |
| 150 - 165 min | 200 Hz     | 4 valid and 1 invalid proxies at 40 Hz |
| 165 - 180 min | unlimited  | 4 valid and 1 invalid proxies at unlimited |

Figure 2. Input frequencies for tests with one certificate; the tests start with 0.1 Hz and changes over 0.3 Hz, 0.5 Hz, 1 Hz, 3 Hz, 5 Hz, 7 Hz, 10 Hz, 40 Hz, 65 Hz and 200 Hz to unlimited.

All performance data is shown in Figure 3. As shown in Figures 3 (a) and (d) the CPU usage rises with the load on the service. For Argus, the CPU usage reaches 100% while the SCAS machine hits 80% usage. The memory usage (Figure 3 (b) and (e)) is not critical but Argus needs more memory. Both services work reliably with no failed authorization in more than 50000 requests. In total the Argus system achieved a higher maximum response frequency of 16 Hz whereas SCAS response saturated at 10 Hz.

4.2. Performance with different proxies
The second test explored the behaviour of the systems when using different proxies for the requests. The test operated with a request frequency of 3 Hz and varied the number of used proxies. Afterwards the frequency was raised to 40 Hz and unlimited. The detailed configuration can be found in column “variation 2” of Table 2. The frequencies are depicted in Figure 4.

The proxy variation includes one invalid proxy, i.e. a not authorised user proxy, which had to be rejected by the services. SCAS shows an interesting effect where the CPU usage drops when using the one invalid proxy (see Figure 5 (a)). This indicates a faster rejection process than a successful authorization. In Argus such a difference cannot be seen.

In this test, Argus achieves a higher response rate than SCAS (see Figure 4), as it did in the first test. The performance data in Figure 5 shows no dependence on the number of certificates used. This might change when using considerably more proxies as expected in production. Here Argus’ feature of caching decisions should be visible.
Figure 3. Server performance data for tests with rising frequency. In the first row the performance data for SCAS is shown. In the second row Argus data is shown.

Figure 4. Input frequencies for tests with variation of used certificates; the first 1 h 45 min the input frequency was held at 3 Hz then it was raised to 5 Hz, 7 Hz, 10 Hz, 40 Hz and unlimited each halted 15 minutes.

Figure 5. Server performance data for tests with certificate variation. In the first row the performance data for SCAS is shown. In the second row Argus data is shown.
5. Pros and cons

5.1. gLExec
gLExec enables logging for payload jobs allowing the sites to implement their security mechanisms for all users. The VO achieves a better data protection for their users. Nevertheless, since the batch system only sees the pilot jobs the site loses the influence on user priorities and can no longer, for example, prioritize national users on national resource shares. After all, VOs can still run user jobs without using gLExec, the sites has no handle to enforce it. With all possibilities of mapping, the gridmapdir remains a single point of failure. At GridKa, gLExec has been in production for more than one and a half years but only DNs from monitoring and test jobs have been mapped. All important functionalities provided by gLExec can be of benefit only if the VOs use it. Since the sites cannot pledge the VOs to use gLExec, this is an issue which can only be met by policies binding the VOs to use gLExec.

5.2. SCAS
The major advantages for SCAS are the simple administration and configuration. The latter is completely covered by YAIM [10], the default configuration tool in gLite. The user banning works the same way as the file transfer protection on the CEs and is easy to integrate into the site security concept. Nevertheless the problematic issue of the gridmapdir being a single point of failure persists. SCAS reaches lower performance than Argus.

5.3. Argus
The main benefit of Argus is the possibility of importing user ban lists from central, regional and local Argus servers. Furthermore it is planned to support more middleware node types, e.g. CREAM CE. The flexible policies ensure a large variability in usage with different services. In comparison to SCAS Argus achieved a better performance. Still there is the increased administrative effort and a suboptimal local banning procedure as well as the need of manual configuration after running the configuration scripts with YAIM. The gridmapdir remains a single point of failure as it is for SCAS. Another minor point in the experience of the author was an early update which broke the service.

6. Conclusion
Argus is an interesting tool for the future as it has the better performance and provides a larger functionality. Nevertheless it has an administration overhead. Therefore the author suggests smaller sites with limited man power to use SCAS and wait for a few more releases before migrating. gLExec has been in production release for a long time, but is still not widely used by VOs. Further tests should cover the performance scaling with rising CPU power, e.g. more cores, and memory. Also the failover mechanisms should be tested for reliability. In the end, the service has to fit into the site concept and the administrators have to be comfortable with the system.

7. References
[1] Gregorio Martinez Parez et al 2011 Fut. Gen. Comp. Sys.: 27 (2011) 40-55
[2] Tadashi Maeno 2008 J. Phys.: Conf. Ser. 119 062036
[3] Alien2 home page: http://alien2.cern.ch/ (Feb 2011)
[4] Security policy for operation of MUPJs - https://edms.cern.ch/document/855383 (Feb 2011)
[5] Argus project: https://twiki.cern.ch/twiki/bin/view/EGEE/AuthorizationFramework (Feb 2011)
[6] Argus load and lifetime tests: https://twiki.cern.ch/twiki/bin/view/EGEE/AuthZLLT (Feb 2011)
[7] SCAS documentation: https://www.nikhef.nl/pub/projects/grid/gridwiki/index.php/SCAS (Feb 2011)
[8] gLExec documentation: http://www.sysadmin.hep.ac.uk/wiki/Glexec (Feb 2011)
[9] gLite Web Home: http://glite.web.cern.ch/ (Feb 2011)
[10] YAIM Web Home: http://yaim.info/ (Feb 2011)