



RAS - JSS TEST REPORT

SOFTWARE TEST AND VALIDATION REPORT

WP4 TASK4 - Verification and Quality Control

Document Filename: **CG-4.4-REP-v1.0-DEMO002-RAS-JSS_TestReport.doc**

Work package: **WP4 TASK4 - Verification and Quality Control**

Partner(s): **NCSR "Demokritos"**

Lead Partner: **LIP**

Config ID: **CG-4.4-REP-v1.0-DEMO002-RAS-JSS**

Document classification: **PUBLIC**

Abstract: This report describes the validation performed on the package RAS-JSS developed by CrossGrid WP 3 tasks 3.1 and 3.2. The tests were performed by Vangelis Floros on behalf of the CrossGrid task 4.4 testbed, verification and quality control. The Job Submission Services (JSS) provide a set of services (exposed as Web Services), which allow performing job submission and monitoring in the CrossGrid Testbed.



Delivery Slip

	Name	Partner	Date	Signature
From				
Verified by				
Approved by				

Document Log

Version	Date	Summary of changes	Author
1-0-DRAFT-A	07/05/2004	Draft version	Vangelis Floros

CONTENTS

1. CONTEXT.....	4
1.1. TEST REQUEST	4
1.2. TEST TEAM	4
1.3. RESOURCES INVOLVED.....	4
2. TEST AND VALIDATION.....	5
2.1. SOFTWARE INSTALLATION.....	5
2.2. ADDITIONAL TESTBED MODIFICATIONS.....	7
2.3. TEST DEVELOPMENTS	7
2.4. USABILITY	7
2.5. FUNCTIONALITY.....	8
2.5.1. <i>Unit tests</i>	8
2.5.2. <i>System tests</i>	8
2.5.3. <i>Stress tests</i>	9
2.6. COMPATIBILITY.....	12
2.7. SECURITY AND NETWORKING.....	12
2.8. PREVIOUSLY REPORTED ISSUES	12
3. ISSUES FOUND.....	13
3.1. ISSUES FOUND IN THE SOFTWARE.....	13
3.1.1. <i>Issue 001</i>	13
3.1.2. <i>Issue 002</i>	13
3.1.3. <i>Issue 003</i>	13
3.1.4. <i>Issue 004</i>	13
3.1.5. <i>Issue 005</i>	13
3.2. ISSUES FOUND IN THE DOCUMENTATION.....	13
3.2.1. <i>Issue 001</i>	13
4. RECOMMENDATION	15
5. REFERENCES	16
6. INTEGRATION/VALIDATION REQUEST	17

1. CONTEXT

Test and validation of the package RAS-JSS (Roaming Access Server – Job Submission Services) developed by CrossGrid WP 3, tasks 3.1 and 3.2. The Job Submission Services are available as web-services and allow users to submit a batch or an interactive job, according to a provided job description (by JDL); cancel a submitted job; get the logging information of their submitted jobs; get the list of their submitted jobs; get the list of available Computing Elements according to their provided requirements.

JSS does not provide any functionality to send input files during the submission operations and retrieve output results of the submitted jobs. These functions are implemented out-of-band by the CrossGrid tools that utilize it (Migrating Desktop, Web Portal and Flood Portal). JSS depends on the WP3.2 Workload package.

1.1. TEST REQUEST

The package test request was assigned by Jorge Gomes (Jorge@lip.pt) to Vangelis Floros (floros@di.uoa.gr) on 06 April 2004. The test requestor was Marco Sottilaro (marco.sottilaro@datamat.it) from Datamat S.A.. Test request was sent to the CrossGrid Test & Validation Team by email in a MS Word document. Although this not the official way, the document contained a request obeying to the official request format (all the fields were present). The contents of the document were inserted in the relevant fields of the CrossGrid Test Request Web Form.

The software URL provided in the “Test Request” did not point to CrossGrid’s repository at GridPortal, rather to the CVS source files area. Currently the software is not yet available in RPM packages. The WP3.2 Workload package, which JSS depends on, is in turn available in RPM files, but the URL that was provided in the test request pointed again to the CVS. The rpms were located in Gridportal repository with the help of Marco Sottilaro

The URL for the Installation, Developer’s and User’s Guide, points to a directory which contains the latest versions of Deliverable 3.5 (in zipped format). To find the relevant information the tester had to go through all sub-documents of the Deliverable and locate the specific parts that concerned RAS-JSS. After request of the Test & Validation team, Datamat compiled a condensed version of the above documents containing only RAS-JSS specific information, which was used as the main reference guide throughout the testing procedure. The file is available from GridPortal under http://savannah.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_1-portals/JSS/docs/cg-wp3.1_3.2-jss-usermanual-042004.doc

1.2. TEST TEAM

The tests were performed by task 4.4 members from “Demokritos”.

- Vangelis Floros (floros@di.uoa.gr)

1.3. RESOURCES INVOLVED

For the purposes of testing the following resources were used in the University of Athens:

- 1 User Interface (cgnode02.di.uoa.gr)
- 1 Workstation running Windows XP, which was used for development and running of the stress tests.

2. TEST AND VALIDATION

2.1. SOFTWARE INSTALLATION

The software was installed in the User Interface of the UoA site which runs *LCG-1.1.4*. All nodes use security patched kernel *2.4.20-30.7.legacysmp*. Packages distributed as RPMs were installed using the LCFGng server. The rest of the packages distributed in *tar.gz* format were installed manually in the UI. All the files have been placed by Datamat in GridPortal's repository under http://savannah.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_1-portals/JSS/tarballs/

For the test and validation procedure three set of packages were required:

- The RAS-JSS server and client packages, distributed in two tar-gzipped files respectively: *cg-wp3.1-jss_server.tar.gz* and *cg-wp3.1-jss_client.tar.gz*. These two files are actually tarballs extracted from the CVS and contain the java sources and relevant scripts required for compilation and deployment of the software.
- A set of WP3.2 Workload packages distributed as RPMs. The list of required RPMs was included in the Test Request document. The RPMs were downloaded from GridPortal under <http://savannah.fzk.de/distribution/crossgrid/autobuilt/i386-rh7.3-gcc3.2.2/wp3/RPMS/>.
- Apache Tomcat 4.1.29 bundled with Axis 1.1 and all the extra jars required by JSS, distributed in tar-gzip format in file *cg-jss-tomcat4.1.29-axis1.1-cogl1.1.tar.gz*.

The above mentioned three tar-gzip files, were copied in the UI end were extracted under the directory */opt/jss* that was created for testing JSS.

The first step was to setup Tomcat. The *CATALINA_HOME* environment variable was set to point to */opt/jss/cg-jss-tomcat* and a new tomcat user with administrator and manager privileges was added. Also the *JAVA_HOME* environment variable was set to point to */usr/java/j2sdk1.4.1_01* (the Java SDK is already present in the UI). The script *\$CATALINA_HOME/bin/startup.sh* was executed in order to start the container which is running at <http://cgnode02.di.uoa.gr:8080>.

JSS server had to be compiled to produce the required classes and jar files. The resulting jar file containing the JSS Web Services is automatically deployed, after compilation, in the Axis servlet running in Tomcat. The compilation and deployment of the software is performed in one step using the script *cg-jss-make-services* which resides under */opt/jss/org/crossgrid/wp3/portals/roamingaccessserver/jobsubmission*.

Compilation is also required for the client-side part of JSS. In order to build the JSS-Client, it is necessary to launch the *cg-jss-make-client* script file in the */opt/jss/org/crossgrid/wp3/portals/portal/jobsubmission* directory. This builds the Java files and the jar library that at the end of the build process is put in */opt/jss/org/crossgrid/wp3/portals/portal/jobsubmission/libs*. It also compiles the simple test clients that can be used to test the functionality of JSS server from the command line. These clients reside under */opt/jss/org/crossgrid/wp3/portals/portal/jobsubmission/webtest* and are the following: *jsstest*, *jsstestcredential*, *jobsubmit*, *jobstatus*, *loginfo*, *jobcancel*, *listmatch* and *userjobs*. The previous cover all of the functionality of JSS server except interactivity, which is not yet completely implemented and was not tested during this procedure.

This jar library can be used in order to create a user client that invokes the Job Submission Services published on the web server.

After deployment the user can run the simple *jsstest* example to verify the correct deployment of the services. In order to be able to submit jobs the WP3.2 Workload packages have to be installed in the same host. The WP3.2 Workload packages are actually replacements of standard packages distributed

by EDG. Due to that the EDG files have to be uninstalled from the UI and be replaced by their Crossgrid counterparts, otherwise rpm complains about package conflicts. The installation was performed centrally from the LCFGng server. A file named *RAS-rpm* was created with the following contents:

```
/* Packages Required by RAS JSS
*/

/* Removing conflicting packages */
-edg-wl-chkpt-api_gcc3_2_2-2.0.18-1
-edg-wl-ui-api-cpp_gcc3_2_2-2.0.18-1
-edg-wl-common-api_gcc3_2_2-2.0.18-1
-edg-wl-logging-api-cpp_gcc3_2_2-2.0.18-1
-edg-wl-logging-api-c_gcc3_2_2-2.0.18-1
-edg-wl-dgas-hlr-jobAuthClient_gcc3_2_2-2.0.18-1
-edg-wl-dgas-hlr-ATMClient_gcc3_2_2-2.0.18-1
-edg-wl-dgas-hlr-ui_gcc3_2_2-2.0.18-1
-edg-wl-logging-api-cpp_gcc3_2_2-2.0.18-1
-edg-wl-logging-api-c_gcc3_2_2-2.0.18-1
-edg-wl-dgas-hlr-jobAuthClient_gcc3_2_2-2.0.18-1
-edg-wl-dgas-hlr-ATMClient_gcc3_2_2-2.0.18-1
-edg-wl-dgas-hlr-ui_gcc3_2_2-2.0.18-1
-edg-wl-logging-api-sh_gcc3_2_2-2.0.18-1
-edg-wl-common-api-java_gcc3_2_2-2.0.18-1
-edg-wl-ui-api-java_gcc3_2_2-2.0.18-1
-edg-wl-ui-gui_gcc3_2_2-2.0.18-1
-edg-wl-ui-cli_gcc3_2_2-2.0.18-1
-edg-wl-bypass_gcc3_2_2-2.5.3-18

/* Packages Required by WP3.2 scheduler */
cg-wp3.2-bypass-2.5.3-18/i486
cg-wp3.2-chkpt-api-2.0.18.2.1-1/i486
cg-wp3.2-common-api-2.0.18.2.1-1/i486
cg-wp3.2-common-api-java-2.0.18.2.1-1/i486
cg-wp3.2-common-api-java-interface-2.0.18.2.1-1/i486
cg-wp3.2-config-2.0.18.2.1-1/i486
cg-wp3.2-dgas-hlr-ATMClient-2.0.18.2.1-1/i486
cg-wp3.2-dgas-hlr-jobAuthClient-2.0.18.2.1-1/i486
cg-wp3.2-dgas-hlr-ui-2.0.18.2.1-1/i486
cg-wp3.2-globus-gridftp-1.5-18/i486
cg-wp3.2-logging-api-c-2.0.18.2.1-1/i486
cg-wp3.2-logging-api-cpp-2.0.18.2.1-1/i486
cg-wp3.2-logging-api-sh-2.0.18.2.1-1/i486
cg-wp3.2-ui-api-cpp-2.0.18.2.1-1/i486
cg-wp3.2-ui-api-java-2.0.18.2.1-1/i486
cg-wp3.2-ui-api-java-interface-2.0.18.2.1-1/i486
cg-wp3.2-ui-cli-2.0.18.2.1-1/i486
cg-wp3.2-ui-config-2.0.18.2.1-1/i486
cg-wp3.2-ui-gui-2.0.18.2.1-1/i486
```

Then *RAS-rpm* was included in *UI-rpm* (the default RPM list file of the UI) and the script */etc/obj/updaterpms* was executed (with the *run* parameter) in the UI host to perform the installation.

In order for the deployed JSS services to be able to use the newly installed packages Tomcat had to be restarted.

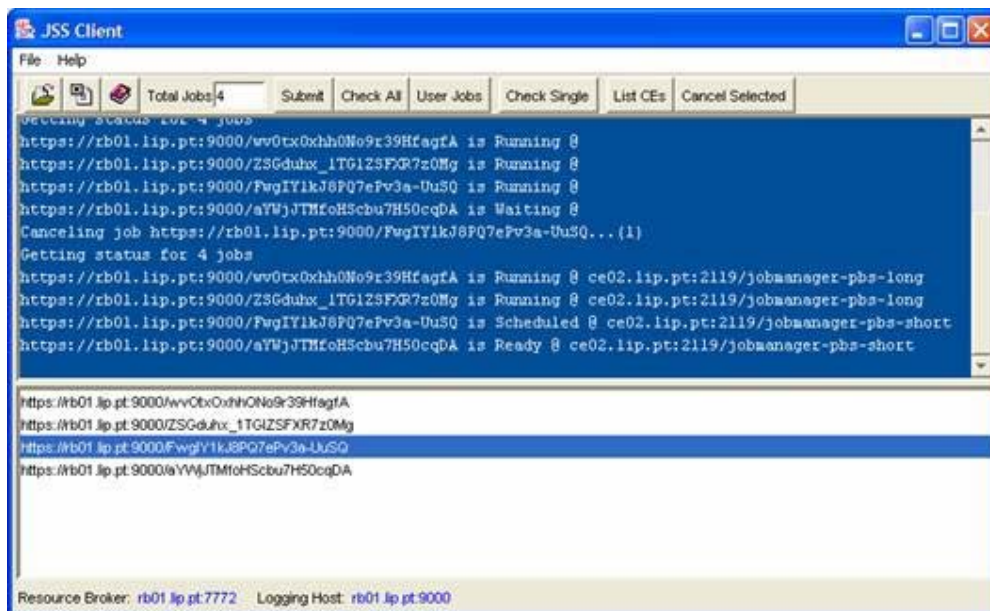
2.2. ADDITIONAL TESTBED MODIFICATIONS

No additional Testbed modifications were required.

2.3. TEST DEVELOPMENTS

A simple graphical client was developed in order to test the WS functionality and to perform the required stress tests. For the development we utilized the JBuilder Java IDE with jsdk-1.4.2 and axis-1.1 toolkit. For the client stub generation, the WSDL of JSS's JobSubmission Web Service was retrieved from the Tomcat server running in the local UI, where JSS was deployed.

The client gives the ability to load the required credentials (proxy certificate) and the JDL file containing the job submission details, and also specify the RB and LB where the job will be submitted and logged respectively. The user can specify the total number of times the job will be submitted to the RB, a facility which is especially useful for the stress tests as it is described below. The user has the ability to retrieve the status of a submitted job (selected from a list), cancel a job, check the status of all the submitted jobs, retrieve the list of CEs that match the submission criteria and also retrieve the list of all jobs that have been submitted in the past by him to the specified RB. A snapshot of the tool is depicted below:



2.4. USABILITY

The software was easy to deploy after all but there were a couple of issues that stalled the procedure. The fact that the packages are not distributed in RPM packages poses a useless deployment overhead for the local administrator. Also from the documentation was not clear (overestimated in fact) what third-party software dependencies existed and the exact number of packages that WP3.2 Workload software is comprised of. The test team was also reluctant to replace the EDG packages with the

relevant WP3.2 Workload once that conflict, but in practice it proved that this replacement did not cause any incompatibilities with the software already installed in the host.

From the programmers point of view the JSS Web Services can be easily utilized. The service operations are straightforward and all the relevant complex parameters required for job submission (like the JDL description or the proxy certificate) are passed as simple byte arrays. Thus it is easy to develop clients that take advantage of the RAS-JSS services.

From the users point of view it is easy to use the simple clients that are provided to submit jobs from the command line. The clients come ready with test input files and provide a simple help facility that makes clear to the user on how to invoke them.

2.5. FUNCTIONALITY

According to the test request the whole provided functionality of RAS-JSS should be tested, except interoperability which is yet in development phase and not fully implemented.

2.5.1. Unit tests

RAS-JSS cooperates with various components in order to provide the job submission and monitoring services, so it makes little sense to perform extensive unit tests, isolating the system from its context. As a simple unit test we can consider the execution of *jsstest* which contacts the service deployed in Tomcat, posts a message, and echoes it back from the service. This test was the first that was performed in order to verify the correct deployment of the Server-side part of JSS and was completed successfully. It does not require a proxy certificate from the user and is not dependent on WP3.2 Workload software. *Jsstest* was also executed successfully from the graphical client since it is provided as an operation by the JSS Web Service.

2.5.2. System tests

For the system tests the rest of the implemented functionality was tested using the provided command line client tools and the developed graphical client. Namely the operations that were tested are:

Jsstestcredential: The operation contacts the Web Service and echoes back a message that the user gives as a parameter. The user's credentials in the form of the proxy certificate are required in order to run this and all of the rest of the tests. The operation completed successfully.

Listmatch: The operation takes as a parameter a JDL file, gets in contact with a defined RB and returns the UIDs of the registered CEs that can execute the defined job. The operation was tested successfully using various JDL files and the two RBs running in FZK and LIP. The response time of the operation depends on the number of CEs that are registered in the RB. In practice it takes more than 10 seconds for the RB in FZK to respond since it currently monitors all the job queues provided in the Crossgrid testbed. The same tests in the LIP RB took only a couple of seconds.

Jobsubmit: The operation submits a job defined in a JDL file. It requires as a parameter the hostname and port of the JSS host, RB host and LB host. The user may define a specific CE where the jobs should be submitted or leave the decision on the RB. On successful submission a JobID string is returned that can be used for monitoring the status of the submitted job. The operation was tested successfully using various JDL files and the RBs and LBs running on FZK and LIP. The time it takes for the client to contact the Web Service and this in turn with the RB is relative fast, since it takes usually 7-8 seconds before the service returns the result of the submission and possibly a JobID (on success) back to the client.

Jobstatus: this operation is used to retrieve the status of a submitted job from the RB. It was successfully used many times during the tests to monitor the execution progress of a large number of

submitted jobs both in the FZK and LIP RB. The response times proved very good since it takes only a couple of seconds for the service to retrieve the status string from the relevant RB.

Loginfo: This operation returns a large set of information regarding a previously submitted job. The user can utilize it to find out the outcome of the execution, reason for failures, target and time of execution etc. It was tested successfully both from the command line client and the graphical client.

Jobcancel: This operation is used to cancel the execution of the submitted job. During the tests the outcome of the execution proved non-deterministic and dependent on the state that the job was when *jobcancel* was invoked. As a result sometimes the RB would immediately respond by cancelling the job, in other cases it would take some time before jobstatus would report that the job was cancelled and in some cases the cancel request was completely ignored and the job execution was carried out completely.

The JSS developers at **Datamat** are familiar with this behaviour. It was explained that from JSS point of view *jobcancel* is just a request of cancelling the job and cannot force it. Before sending the request to the RB, *JSS-jobcancel* checks whether the status of the job allows the operation. Some job states in fact don't allow performing it. For instance, it is not possible to cancel a job that is in DONE state. In this case the JSS client gets an exception message *"the state of the job doesn't allow cancelling of the job"*.

Userjobs: This operation returns a complete list of the jobs that have been submitted in the past by the current user to a given RB. The first tests, of the operation, that were performed failed by throwing the following java exception: *"unable to retrieve the list of the jobs"*. According to **Datamat** this is a known issue that has to do with incompatibility problems of the LCG-1 version of RB. Indeed running the equivalent EDG command *"edg-job-status -all"* from the UI we got the same results. The issue was fixed in the LIP RB (which runs LCG-2) during the period of testing and an update version of JobSubmission service was provided by Datamat, which cooperated well with the RB. Using these modifications we could successfully run the *userjobs* operation with the RB at LIP.

2.5.3. Stress tests

For the stress tests the graphical client that was described in section 2.3 was used. Two different stress tests were performed. One was the submission of 100 jobs from a single client to the FZK RB using jobsubmission operation and the second was submission of 100 jobs from 10 different clients simultaneously (10 jobs per client).

Test 1

100 jobs were submitted, from a single client, one after the other. The total time that it took for all the jobs to be submitted was 17 minutes and 24 seconds. During the submission phase we monitored the CPU and memory load of the UI host were JSS runs. The peek CPU load and Memory usage experienced are summarized in the following table:

Peek CPU load	0.28
Peek Memory Usage	251660 K
Peek Swap Usage	5672 K
Max Axis Sessions	107

Notice that the UI is a Pentium 4 2GHz single CPU machine with 256MB main memory. No other users were connected during the tests. The services that UI hosts are the typical minimum services allows in the CrossGrid Testbed.

The following table summarizes the instant CPU load and number of Axis Sessions during the submission of the jobs:

Time (min:sec)	CPU Load	Axis Sessions
4:30	0.05	34
5:00	0.03	39
6:00	0.04	45
7:00	0.01	51
8:00	0.07	56
9:00	0.02	62
10:00	0.01	68
11:00	0.22	74
12:00	0.09	79
13:00	0.03	84
15:00	0.00	94
16:00	0.11	100
17:24	0.08	107

As it can be seen in the above table the load of the system was kept relative low and no service response degradation was experienced during the test. All the jobs were successfully submitted to the RB. Their status was retrieved with no problems using *Jobstatus* operation. The final outcome of the jobs is irrelevant of JSS service and dependent on the RB. In fact during the test the RB at FZK experienced problems which prevented from the correct delivery of the jobs to the selected CE's. We do not feel that this should be considered a problem of JSS since its duties are completed when RB has accepted the job for submission.

Test 2

10 instances of the graphical client were used to submit 100 jobs, distributed evenly in every client (10 each) from the same host. The RB in FZK was again selected as the target RB. Unfortunately the CrossGrid infrastructure was not able to handle this synchronous job submission and most of the jobs failed to submit successfully. The table bellows summarizes the test results per client:

Client No.	Successful submitted jobs	Result
1	10	Success
2	9	Failed with java exception
3	1	Failed with java exception

4	3	Failed with java exception
5	0	Failed with java exception
6	5	Failed with java exception
7	4	Failed with java exception
8	0	Failed with java exception
9	0	Failed with java exception
10	1	Failed with java exception

From the above clients 2,3,4,5,6,7,9 and 10 failed by throwing the following java exception:

```
AxisFault
faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.userException
faultString: java.rmi.RemoteException: unable to submit the job: Unable to submit
the job: Unable to connect to remote (rb.fzk.de:7772)
faultActor: null
faultDetail:
stackTrace: AxisFault
faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.userException
faultString: java.rmi.RemoteException: unable to submit the job: Unable to submit
the job: Unable to connect to remote (rb.fzk.de:7772)
faultActor: null
faultDetail:
```

Client 8 on the other hand threw the following exception

```
AxisFault
faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.userException
faultString: java.rmi.RemoteException: unable to submit the job: Failed to
establish security context...
faultActor: null
faultDetail:
stackTrace: AxisFault
faultCode: {http://schemas.xmlsoap.org/soap/envelope/}Server.userException
faultString: java.rmi.RemoteException: unable to submit the job: Failed to
establish security context...
faultActor: null
faultDetail:
```

According to Datamat both "Unable to connect to remote (rb.fzk.de:7772)" and "Failed to establish security context..." are messages coming from the RB server. In all cases JSS was able to process the submission request, but when RAS tried to contact the specified RB:

- in the first case the RB wasn't available (probably due to too much stress)
- in the second case the RB was contacted, but something went wrong during X509 credential check.

It is quite easy to get the second error message and has caused problems to the RB in the past. The problem is already known to the developers and in fact it is mentioned in the Test Request form: *"Some problems with the user credential could rise when the server side has to handle more than a service request at the same time. It has already been fixed for the next release. It isn't possible to fixed it by the current release."*

2.6. COMPATIBILITY

During the tests two compatibility issues were raised. The first concerns the *JSS-userjobs* operation's interoperability problem with the LCG-1 RB that is currently installed in FZK. As it was reported by the administrators at LIP and FZK, LCG-1 RB has a lot of problems in the logging a bookkeeping and may require quite an effort in order to support JSS in its full extent. It seems preferable to wait for the deployment of LCG-2 in the CrossGrid Testbed and then deal with any remaining compatibility issues.

The second issue concerns the fact that WP3.2 Workload packages, that RAS-JSS depends on, have to replace existing packages distributed with LCG-1. During the tests no compatibility problems were experienced by using these packages which seem to provide backward compatible functionality with their LCG-1 counterparts, but it is an issue that has to be verified, probably by the testing and validation process of WP3.2 Workload software itself.

2.7. SECURITY AND NETWORKING

Since JSS services are deployed on Tomcat TCP port 8080 should be open for inbound connections in order for remote clients to be able to invoke them.

Users are identified by their proxy certificates. Although this is the correct approach for authentication the fact that the user credentials are transmitted over an insecure channel to Tomcat, poses an important security risk. In practice an adversary can easily eavesdrop the communication between the client and the web server and steal the credentials. This in turn will give him/her the ability to exploit the Testbed resources, or even open a remote terminal on a CrossGrid host (using *glogin* for instance), at least till the proxy expires. This issue has been identified by the RAS-JSS programmers who are planning to provide secure channels in the next release of JSS.

2.8. PREVIOUSLY REPORTED ISSUES

This was the first version of RAS-JSS that was tested.

3. ISSUES FOUND

3.1. ISSUES FOUND IN THE SOFTWARE

3.1.1. Issue 001

(Severity: high Priority: immediate)

The installation process of RAS-JSS as it currently is requires from the administrator to manually transfer the application tarballs in the target host, extract them, compile the source code and configure Tomcat. This procedure does not comply by no means with the centralized administration scheme that CrossGrid has adopted by using LCFGng and should be fixed as soon as possible. For this reason it is required the close cooperation between all parties involved, namely the JSS developers and the CVS/autobuild administrators, so that the autobuild procedure can produce the necessary RPM packages.

3.1.2. Issue 002

(Severity: high Priority: medium)

JSS may throw a "*Failed to establish security context...*" java exception when submitting multiple jobs simultaneously. According to Datamat this bug is already fixed in the next release of the software.

3.1.3. Issue 003

(Severity: high Priority: medium)

JSS may throw an "*Unable to connect to remote (...)*" java exception when submitting multiple jobs simultaneously. It should be verified that this is not a problem of the software itself rather of the Resource Broker. It should be noted that this problem appears both in the FZK (LCG-1) RB and the LIP (LCG-2) RB.

3.1.4. Issue 004

(Severity: medium Priority: medium)

JSS should use a secure channel between client and Tomcat so that the user credentials can be transferred securely. The lack of this security feature may impair the security of CrossGrid's Grid Infrastructure. According to Datamat this will be supported in the next release of JSS.

3.1.5. Issue 005

(Severity: medium Priority: low)

JSS-userjobs operation is incompatible with the LCG-1 Resource Broker and the LCG-2 RB requires a fix to work. This issue is expected to be resolved when the CrossGrid Testbed migrates to LCG-2.

3.2. ISSUES FOUND IN THE DOCUMENTATION

3.2.1. Issue 001

(Severity: medium Priority: medium)

The installation/user manual that was provided during the testing period proved very helpful and certainly more useful than the deliverables that were suggested at first as reference material. Nevertheless it requires extensive restructuring and fixing in order to be improved in terms of

consistence and correctness. The developers of RAS-JSS should improve it and update it with new features of the upcoming release.

4. RECOMMENDATION

Apart from the issues that were discussed in the previous sections we would also like to bring to notice of the following subjects:

The usage of the User Interface machine for hosting the JSS Web Services should be considered a rough and temporary solution. Ideally, in a production environment, a dedicated machine should be used for hosting the service. If there is a feeling that the EDG scheme is already overloaded with different machine roles, JSS should at least be co-hosted in a role which already runs server applications preferably Apache Tomcat itself.

JSS is currently in a state that provides useful functionality, leveraging the Web Services technology providing an open interface of the LCG-1 Grid middleware. Nevertheless it is important in the context of CrossGrid this functionality to be enriched with interactivity characteristics which of course are expected to be available soon by JSS. In the next months of the project the development team should concentrate their efforts towards this area so that we can have a good level of interactivity support from JSS.

Lastly as a general commend (beyond CrossGrid) it would be interesting to reconsider the issue of input and output data file staging which has been currently resolved from the relying software (Migrating Desktop, Web Portal) by using ad-hoc / out-of-band mechanisms. A unified service for file staging, job submission and monitoring could probably provide a functionally enhanced solution to the problem.

5. REFERENCES

- [1].http://savannah.fzk.de/cgi-bin/viewcvs.cgi/crossgrid/crossgrid/wp3/wp3_1-portals/JSS/docs/cg-wp3.1_3.2-jss-usermanual-042004.doc
- [2].<http://wp3.crossgrid.org/doc/d35-10-02-2004/CG3.0-D3.5-v1.0-PSNC010-Proto2Status.doc>
- [3].http://www.lip.pt/computing/projects/crossgrid/task4/softvalidation/108124927389.5035578011633/request_form.html

6. INTEGRATION/VALIDATION REQUEST

Append HERE the integration/validation request.