Table of Contents

EGI Repository Design Document......1
- Executive Summary......1
- Glossary......1
- Table of Contents......1
- Introduction and Requirements and Objectives......1
  - Operations on the repository......2
  - Contents of the repository......2
  - Supported Projects......2
- Users and User's Roles in the Repository......2
  - Repository Contents for End Users......3
  - Repository Administrators......3
- Use Cases......3
  - The EMI Software Use case (fully supported software)......3
  - Associate software use case......4
  - User Community Software use case......5
  - Administrator's Access use case......5
  - Download of Components from end users......5
  - Workflow Support......6
    - State transitions of the releases......6
    - Interaction with the tools that supports the verification process......8
    - YUM/APT related metadata provision......8
- User management and Security......9
  - Alternatives......9
    - Certificate based access......9
    - EGI SSO access......9
  - Proposed Solution......9
- MetaData Definition......9
  - All Software Components......9
  - Real Software Components......10
  - Virtual Software Components......10
- Access Interfaces......10
  - For Software Providers......10
  - For end users......10
- Interfaces with External Tools......10
- Web Front End Description......11
- Database......11
- Other Considerations......11
  - Package Signing......11
    - RPM......11
    - DEB......11
  - Issues to consider......12
    - Private key protection......12
    - Life time of the key......12
    - Key distribution......12
    - Software Packages......12
- Accounting......12
- Monitoring......12

EGI Repository Design Document

Executive Summary

This document provides the current (high-level design of the EGI repository. We try to visualize how the repository will be exploited by the European and international research community to cater for the needs and expectations of the EGI community. This is a first draft for discussion before the middleware meeting at CERN on the 6th, 7th of Aprli.

Glossary

Table of Contents

Introduction and Requirements and Objectives

The EGI Software Repository will be a highly available source of software components available for inclusion in the UMD Release or for direct use by NGIs. The repository will have:

- real components (with source and binary releases physically located on the servers) or
- virtual components (with source and binary releases located on the software provider¡s servers). Components (and their specific releases eventually) in the repository are expected to fall into the following categories:

UMD Components, included in UMD distribution; they are further classified as:

- fully supported – terms of support (SLD) were negotiated with the component provider, and they are documented and published with the component
- community supported – there is no explicit support agreement, however, the component is of general importance, it is actively used by the community and supported by its provider associated components: software that is generally recognized to work well with UMD, however, it is not its intrinsic part, and EGI provides no explicit support to it, leaving it to the interested community (similar to the existing RESPECT program in EGEE)

The repository will contain versioned binary (for multiple supported platforms, in a format native for the platform, e.g. RPM, deb etc.) components as uploaded by software providers. Access to the source code must be provided unless an exemption has been specifically negotiated with the provider. This can be achieved by explicitly uploading the source or by using an established repository (e.g. SourceForge). Versioned components will appear in one of the states of their life cycle (contributed, under evaluation, in staged rollout, in production, rejected, and deprecated) according to their state w.r.t. the software release process described above. Components will be arranged into installation repositories for automatic download through existing platform-specific repository formats (e.g. yum, apt, etc.). Multiple such repositories per platform will be available, depending on the state of the component (pre-release, released, in-production). Only components in the in-production state are exposed in repositories used for automatic updates.

To enable effective interaction between the teams within EGI and the external software provider various support tools will be provided. These will include issue tracking systems (e.g. Bugzilla, Savannah, ...), source code repositories (eg. tar balls source rpms), wikis and mailing lists. The specific systems will be chose during the project preparation phase.

Operations on the repository

The following operations will be support by the repository:

1. Download the software packages directly from the repository using yum/apt.
2. Sync repository contents using rsync.
3. Add/remove users
4. Add/remove groups
5. Add/remove software components
6. Add/Edit software component metadata
7. Create/delete yum/apt sub-repositories
8. Approve/reject software components
9. Digitally sign software packages (to be decided)

Contents of the repository

The UMD Components, included in UMD distribution; they are further classified as:

- fully supported – terms of support (SLD) were negotiated with the component provider, and they are documented and published with the component
- community supported – there is no explicit support agreement, however, the component is of general importance, it is actively used by the community and supported by its provider
- associated components: software that is generally recognized to work well with UMD, however, it is not its intrinsic part, and EGI provides no explicit support to it, leaving it to the interested community (similar to the existing RESPECT program in EGEE)

The repository will contain versioned binary (for multiple supported platforms, in a format native for the platform, e.g. RPM, deb etc.) components as uploaded by software providers. This can be achieved by explicitly uploading the component or by using an established repository (e.g. SourceForge). Versioned components will appear in one of the states of their life cycle according to their state. Based on the metadata provided by the software providers, the components will be arranged into installation repositories for automatic download through existing platform-specific repository formats (e.g. yum, apt, etc.). Multiple such repositories per platform will be available,

Supported Projects

At the moment the foreseen users of the repository (supported projects) are:

- EMI (Fully Supported)
- lcg-CA (Fully Supported)
- StratusLab (We assume that the initial status will be associated component)
- EDGI (We assume that the initial status will be associated component)
- EGI ? (We assume that the initial status will be associated component)
- Various Respect Projects (Associated Component)

Users and User's Roles in the Repository

There are four type of users accessing the services provided by the UMD Repository:

- **End Users**: They have read-only access to the Repository services and they can browse all public web pages and the production and staged-rollout package repositories. End users are not authenticated and considered as anonymous users to the system.

- **Software Providers**: Software providers should have RW access to specific parts of the UMD Repository Services. They can edit only information regarding the software they "own". Access to the programmatic interface
- **Middleware Verification Unit**: The middleware verification unit should have RW access to specific parts of the UMD repository services. They can edit the information on any software package
- **Repository Administrators**: The repository administrators have full administrator access to the repository in order to perform the repository maintenance tasks.

Repository Contents for End Users

The contents of the repository for the End Users are:

- Binary and Source RPMS (RPM, SRPM)
    - ♦ Delivery Methods: YUM, APT, http, rsync
- Tarballs
    - ♦ Delivery Methods: http, rsync
- DEB
    - ♦ Delivery Methods: APT, http, rsync

Repository Administrators

The administrators of the UMD repository will be able to perform restricted operations on the contents of the repository, such as, checks whether the software were downloaded correctly or not, physically moves the data between the repositories (based on the metadata set and upon the state change as this is set by the Evaluators/Verifiers), create YUM/APT sub-repositories (based on the metadata) e.t.c.

Use Cases

The EMI Software Use case (fully supported software)

Goal: The EMI software provider contributes a new release of the software component to the UMD repository. The release is moving forward throughout each face and it is finally populated to the production repository.

NOTE: The EMI software provider actually provides the corresponding release-based metadata including a URL where the contributed software release resides (an rsync URL would be the most suitable). Then, it is of responsibility of the UMD repo to download the release from the specified URL (PULL mode).

Primary Actor: EMI software provider, Evaluators/Verifiers

Scenario:

1. The EMI software provider makes sure that his/her packaged release/component meets all the agreed criteria.

2. The EMI software provider logins into the UMD repo web interface

3. Inserts the necessary metadata information associated with the release (a list of proposed metadata it is provided at paragraph [tbs])

4. Based on the URL provided by the EMI software provider, the release is downloaded to a preliminary area of the UMD repository.

5. A set of checks are performed in order to find whether the submitted release has been downloaded correctly, or not.

6. If no problem arises, the newly contributed software release it is moved into the /scratch sub-repository of the UMD repo.

7. The evaluators get notified that a new release of the EMI software has just been submitted, it is in the /scratch sub-repository and it is ready to be reviewed.

8. The verification/reviewing process begins.

9. If the release complies with the verification criteria, the evaluators/verifiers sets the state of the release as "Verified-InStageRollout"

10. The new release it is moved to the /stage-rollout repository and based on the metadata, the appropriate YUM/APT repositories are constructed.

11. The evaluators get notified that a new release of the EMI software has just been moved/released into the stage-rollout repo and it is awaiting for approval in order to go into the /production repo.

12. Once more, the verification/reviewing process begins.

13. If the release complies with the verification criteria, the evaluators/verifiers sets the state of the release as "Verified-InProduction"

14. The new release it is moved/released into the /production repository and based on the metadata, the appropriate YUM/APT repositories are constructed.

Associate software use case

Goal: The software provider contributes a new release of the associate software component to the UMD repository.

Primary Actor: Associate software provider, Evaluators/Verifiers

Scenario:

1. The software provider makes sure that his/her packaged release/component meets all the agreed criteria.

2. The software provider logins into the UMD repo web interface

3. Inserts the necessary metadata information associated with the release (a list of proposed metadata it is provided at paragraph [tbs])

4. Uploads the software component

5. A set of checks are performed in order to find whether the submitted release has been uploaded correctly, or not.

6. If no problem arises, the newly contributed software release it is moved into the /scratch sub-repository of the UMD repo.

7. The evaluators get notified that a new release of the software has just been submitted, it is in the /scratch sub-repository and it is ready to be reviewed.

8. The verification/reviewing process begins (in this case we assume that at least a simple verification process should be followed i.e. to check the content/formality of the provided release notes e.t.c.).

9. If the release complies with the verification criteria, the evaluators/verifiers sets the state of the release as “Verified-InProduction”

10. The new release it is moved/released into the /production repository.

User Community Software use case

Goal: The software provider contributes a new release of the associate software component to the UMD repository. There is no verification procedure; all new packages are added to a production repository.

Primary Actor: Community Software provider

Scenario:

1. The software provider makes sure that his/her packaged release/component meets all the agreed criteria.

2. The software provider logins into the UMD repo portal

3. Inserts the necessary metadata information associated with the release (a list of proposed metadata it is provided at paragraph [tbs])

4. Uploads the software component

5. A set of checks are performed in order to find whether the submitted release has been uploaded correctly, or not.

6. If no problem arises, the newly contributed software release it is moved into the /scratch sub-repository of the UMD repo.

7. The new release it is moved/released into the /production repository.

Administrator's Access use case

Goal: Accessing the repository in order to perform restricted operations on the contents of the repository, such as, checks whether the software were downloaded correctly or not, physically moves the data between the repositories (based on the metadata set and upon the state change as this is set by the Evaluators/Verifiers), create YUM/APT sub-repositories (based on the metadata) e.t.c.

Primary Actor: Administrators.

Scenario:

1. Administrator logs into his/her privileged account

2. He/she is able to perform a set of administrative operations.

Download of Components from end users

Downloading packages from the UMD can be achieved in two ways:

1.Using the repositories through yum/apt (**RECOMMENDED**).

2.Downloading the packages from the UMD portal.

In the first case, the user adds the repository and certificates to its repository management system. This ensures timely updates and automates the procedure. Some users might find it handy to download certain packages by hand from the portal itself. Such an approach may be used by end users to revert their installation to a previous version or to get a particular deprecated version due to compatibility concerns.

3. rsync will be supported for repository mirroring

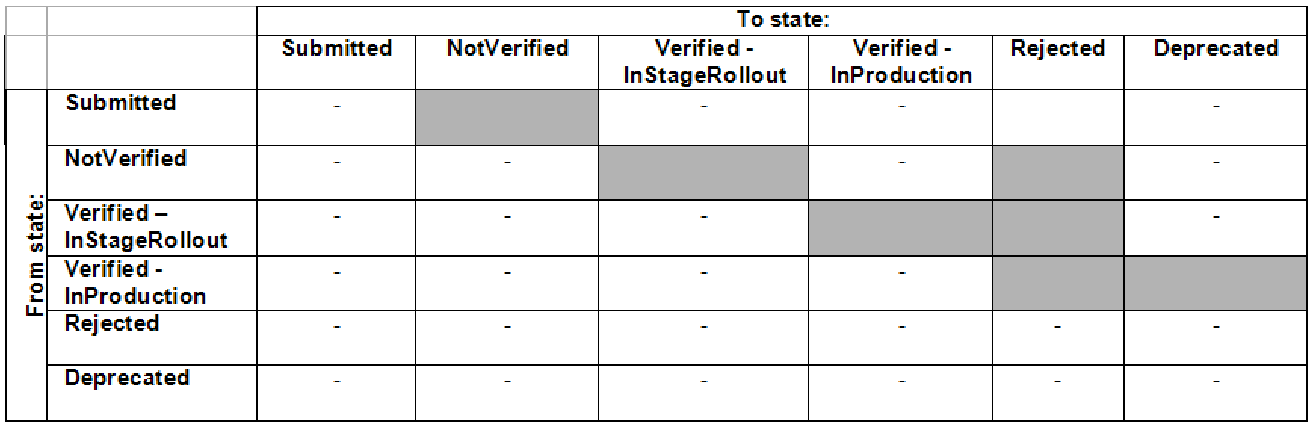
Workflow Support

The framework and the tools that will be finally, selected/created in order to cover the UMD repository workflow, should satisfy the following high-level requirements:

- State transitions of the releases
- Interaction with the tools that supports the verification process
- YUM/APT related metadata provision

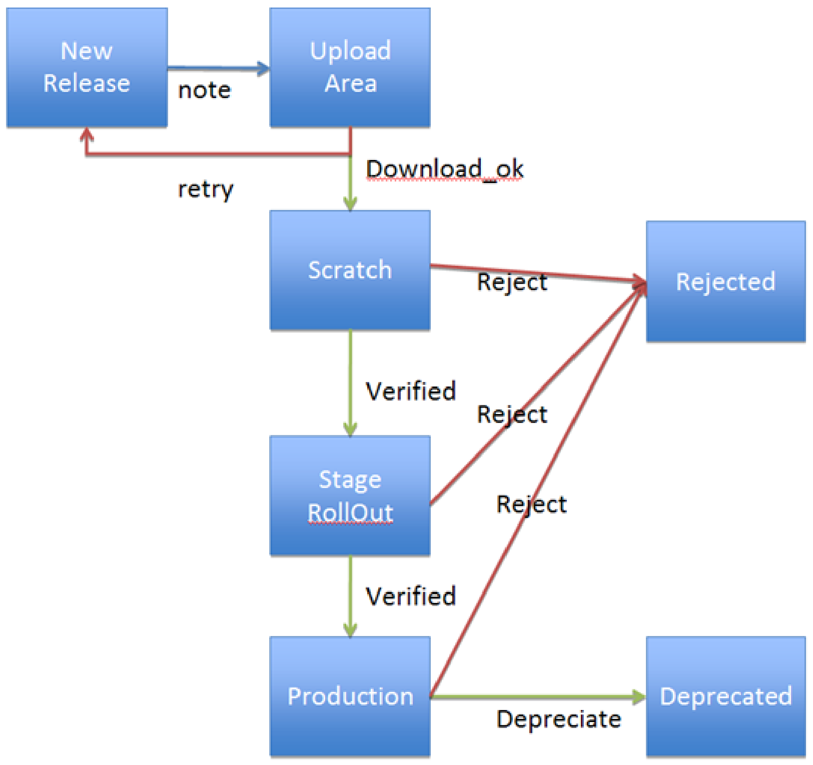
State transitions of the releases

For the fully supported software the following states should be addressed and the corresponding transitions should be satisfied, for every release:

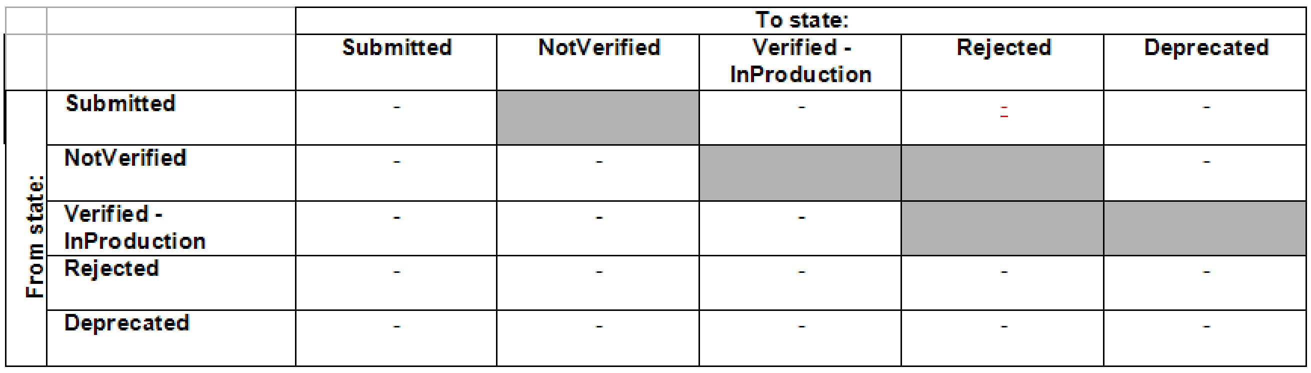


		To state:					
		Submitted	**NotVerified**	**Verified - InStageRollout**	**Verified - InProduction**	**Rejected**	**Deprecated**
From state:	**Submitted**	-	▨	-	-		-
	NotVerified	-	-	▨	-	▨	-
	Verified – InStageRollout	-	-	-	▨	▨	-
	Verified - InProduction	-	-	-	-	▨	▨
	Rejected	-	-	-	-	-	-
	Deprecated	-	-	-	-	-	-

And in a graphical representation:

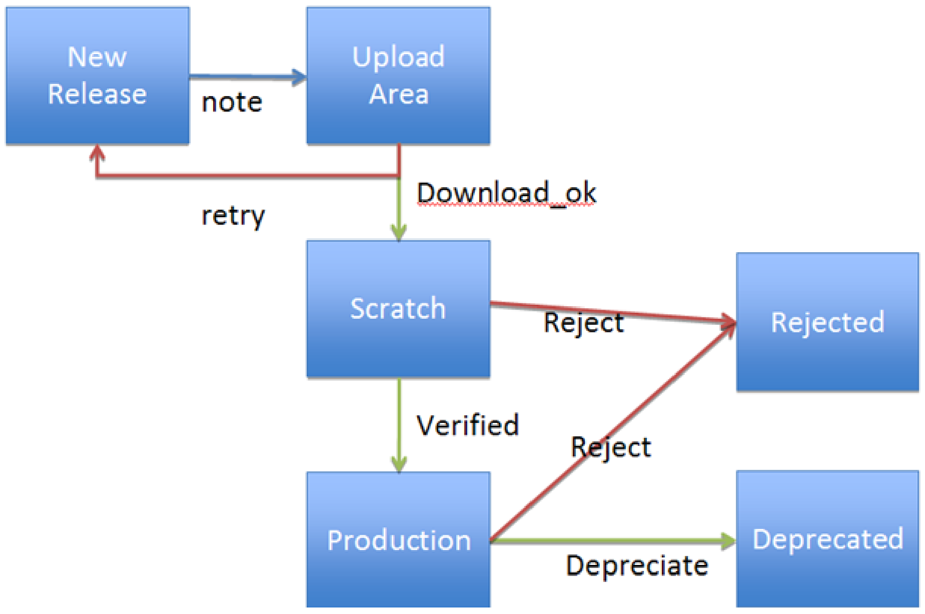


As far as the associate and the community software concerns (paragraphs [tbs] and [tbs]), the following states should be addressed and the corresponding transitions should be also satisfied, for every release:



		To state:				
		Submitted	NotVerified	Verified - InProduction	Rejected	Deprecated
From state:	Submitted	-	▨	-	-	-
	NotVerified	-	-	▨	▨	-
	Verified - InProduction	-	-	-	▨	▨
	Rejected	-	-	-	-	-
	Deprecated	-	-	-	-	-

Bellow you can find, a graphical representation of the information contained in the previous table.



In both, the pre-mentioned, cases there are points that need extra clarifications:

- A software release can be moved only forward, meaning:

```
Submitted -> NotVerified -> [ Verified-InStageRollout ] -> Verified-InProduction -> Depreciated
```

- Regardless release¢s current state, it could be moved into “Rejected” state by the Evaluators/Verifiers. The “Rejected” state it¢s a permanent/final state and when the software provider “fixes” the problem, the software release has to be re-submitted into the UMD repo (a new revision number will be assigned to it).

Interaction with the tools that supports the verification process

There is a need for the UMD repo to access the state of each release as this is set by the Evaluators/Verifiers, thus an interface to the tool(s) that covers the verification process should be available.

YUM/APT related metadata provision

The corresponding metadata needed to be provided by the software provider for the creation of the YUM/APT related sub-repositories. This metadata main aim is to give answers to the following queries:

- Does the software provider wants his software product to contain YUM and/or APT sub-repositories?

- In case of YUM;
    - ♦ Does he wants metapackage based YUM or group-based (group based is currently the case of glite-UI and glite-WN in glite-3.2)?
    - ♦ In case of metapackage-based YUM;
        - ◊ Are all the required metapackages contained in the submitted release?
        - ◊ Did he provide a list of the identified metapackages?
    - ♦ In case of group-based YUM;
        - ◊ Did he provide the corresponding list (possibly in an xml format – i.e. a comps.xml file) that contains all the indentified groups?

- In case of APT;
    - ♦ Did the software provider specified under which directory he wants the APT repository to be created?

User management and Security

There are the following requirements for the User Management:

- Take advantage of existing authentication services available in EGI
- User accounts should be able to be members in Groups and be assigned specific roles
    - One group for the people belonging to the Middleware Verification Unit
    - One group for the Repository Administrators
    - One group for each of the Software Providers (i.e. EMI, lcg-CA etc) that will be registered in the UMD repository
    - In order to ease the administration burden there should be the role of Group Admin for each group in order for the person who is assigned this roles to be able to perform Group administration tasks (e.g. adding or removing users) internal to the group.

Alternatives

Certificate based access

Most of the people collaborating with EGI have or are able to acquire X509 certificates from an IGTF accredited certification authority. Using X509 certificates to access web based services is a method commonly used within EGEE/EGI and there is substantial technical know-how in implementing such a solution.

All users requiring RW access to the repository will be members to a specific VO (e.g. umd-repo.vo.egi.eu) and will be members to the Group they belong. VOMS provides the facilities for setting up such groups and assigning roles.

EGI SSO access

CESNET will be offering an EGI wide SSO service. We could take advantage of this services and use the accounts provided by the EGI SSO service in order to authenticate the people who require RW access to the repository services.

All users requiring RW access to the repository will be members to at least the Group they belong. This means that the EGI SSO should allow users to be grouped into groups.

According to our current understanding the existing EGI SSO is not based on one of the commonly used Federation protocols, but instead it will be an LDAP holding all the user accounts. Is this true?

Proposed Solution

It is seems that certificate based access is the best choice for this type of service. EGI collaborators will not have to create an EGI account in order to access the services and they will be able to register only at the specific VO for the UMD repo.

MetaData Definition

All Software Components

- Static (Rarely Change)
    - Software_Provider
    - Software_Component
    - Creator
    - Contact

    - Description
    - SLD_Type (fully supported, community supported, associated components)

Real Software Components

- Dynamic (Change at every x.y. release)
    - Date
    - Version
    - Release_Notes_Url
    - Services_Affected {set}
    - Change_Log
    - Rsync_Url (host and path from which we pull the data)
    - Distribution_method {yum,apt,tar)
        - Dependencies
        - Service_Groups (for more details please refer to section [[YUM/APT related metadata provision][]])
    - Documentation_Url

Virtual Software Components

- Dynamic (Change at every x.y. release)
    - Date
    - Version
    - Release_Notes_Url
    - Change_Log_Url
    - Repository_Url (URL from where the software packages can be downloaded)
    - Documentation_Url

Access Interfaces

There is a single web portal for software providers and users alike. This provides consistency between the available information for each group. The repository portal provides an authentication mechanism for software providers, while end users can browse the site for releases/packages etc.

For Software Providers

Software providers will be given access to specific parts of the portal pertaining to their role in the process (e.g. software provider, Middleware Verification Unit, Repository administrator). Software providers have personal accounts and site permissions, subject to their group policy.

The procedure for software provider access will be supported by the respective authorization mechanism that will be chosen. i.e, VOMS, or EGI SSO (See section User management and Security)

For end users

End users will be able to search and download releases or specific packages. A method for following news will be provided (e.g. RSS feed, e-mail notifications), either for the whole repositories, specific releases or individual packages.

Interfaces with External Tools

A batch update method will be available. A specific XML document, containing all the necessary metadata can be uploaded, that can contain any number of packages organized per release number.

Web Front End Description

The web frontend contains news for new releases, links to the EGI sites, documentation for the procedures required of end users and software providers to use the portal and repositories, a search interface for packages/releases and a login form for the software provider/Middleware verification team/Repository administrator, interface hidden to end users.

Database

There will be one database that will hold all the metadata of the software components. The database will be replicated across three HellasGrid sites in a master - slave scheme. The database is a key component for the web frontend provided to the software providers, for the web frontend provided to the Middleware Verification Unit and for the repository maintenance tools that will build the repositories based on the metadata information.

Other Considerations

Package Signing

Both RPM and DEB package formats support digital signatures using GnuPG .

The reason for signing a package is to provide authentication. With a signed package, it's possible for the user community to verify that the package they have was in our possession at some time and has not been changed since then. That "not changed" part is also a good reason to sign your packages, as digital signatures are a very robust way to guard against any modifications to the package.

The RPM package format supports also the notion of adding more than one signature to a package. This way we can provide a means of documenting the path of ownership from the package builder to the end-user.

As an example, the software provider creates a package and signs it with the their key. The UMD Verification Unit checks the signature and adds the software to the repository, in essence stating that the signed package received by them is authentic. Upon successful certification of the software package, the package is signed again by the key of the UMD Repository and it is moved into the production repository.

The package now makes its way to the end users that wish to deploy the package. After checking every signature on the package, they know that it is an authentic copy which has gone through the EGI verification process.

RPM

The support of digital signatures in the RPM package format seems to be more mature that in the DEB format. Each RPM package is digitally signed separately and can have multiple signatures.

DEB

The DEB format support digital signatures, but this is not widely used. Instead the preferred way is to digitally sign the Release file which holds the md5 checksums for all the software packages. The DEB package format supports the notion of different type of signatures. In predefined steps during the verification process new type of signature is added to the package. (There can be only one signature per type)

Issues to consider

Private key protection

The private key used for the digital signing should be properly protected and not reside on a publicly accessible system.

Life time of the key

By default GnuPG creates keys without expiration date. The current best practice is to create keys with limited life time and re-key at well defined intervals. The question of what happens to the packages that have been signed by an expired key needs to be investigated.

Key distribution

In order for the users to be able to verify the digital signatures, it is necessary that the public key is distributed before hand to them in a secure way. Although we have extensive experience in distributing trust anchors by the EUGridPMA , the need of distributing public keys used for the verification of software package signatures might result into a chicken & egg problem.

Software Packages

When a software package is signed, then the package contents change in order to include also the signature(s). This mean that we will have to keep both the original package as it was uploaded by the software provider and the final signed package. Another side effect is that the checksum of the package will also change, so we will need to generate and maintain the checksum files.

Accounting

The following information will be provided for accounting purposes:

- Number of Software providers *
- Number of Software Components (Real/Virtual) (Registered/Under Verification/Approved/Rejected)*
- Number of software components verified per verifier *
- Number of downloads *
- Duration of the verification process for each component *
- Availability of services *

Monitoring

All the UMD Repository Services will be monitored by the HellasGrid monitoring infrastructure based which is based on Nagios. There will be monitoring probes for all the UMD Repository Service components.

Currently the following UMD Repository Service components have been identified:

1. The yum/apt repositories. They will be used by many sites for installing the software. It is expected that they will be continuous use of the services, with spike whenever a new UMD release is rolled-out. A failure will impact a large number of users/sites. HIGH CRITICALITY
2. The web front end for the end users. This is the front end from which the end users will be able to read information regarding the EGI software releases. It is expected that they will be continuous use of the services, with spike whenever a new UMD release is rolled-out of course not at the same levels as the yum/apt repositories. A failure especially during a roll-out process will impact a large number of users and will damage the public image of the EGI. MEDIUM CRITICALITY

3. The web front end and programmatic access interface for the software providers. These are the services used by the software providers in order to submit their new software releases to the UMD repository. The service it is expected to be used by the software providers at intervals. A failure of the service will affect only the software providers. MEDIUM CRITICALITY
4. The web front end from the Middleware Verification Unit. This is the front end that will be used by the Middlware Verification Unit in order to approve or reject a software release. A failure of the service will affect the time it takes for a new software release to be added in the production repository. LOW CRITICALITY
5. The backend database. This is used by all the web front-ends and the repository maintenance tools. If there is a failure in the database, then all the afore-mentioned services will be affected. HIGH CRITICALITY.

The package repositories, the web front ends and the databases will be replicated across three HellasGrid sites. For the package repositories and the web front ends there will round robin (RR) DNS entries. The monitoring probes will monitor the availability of all the services and if any one fails it will be automatically removed from the RR. This mechanism has been used effectively for more than 3 years on the HellasGrid infrastructure for providing high availability for the WMS and the top level BDII. In the case of the database, if the master instance experiences a failure, one of the slave instances will be promoted to master.

This topic: HellasGrid/EGIUMDRepository > DesignDocument
Topic revision: r21 - 2010-03-30 - 15:42:01 - IoannisLiabotis