LATTE Security Architecture – User Rights Management

Type:	Design Document (Draft)
Version:	V0-0-2
Author:	S. Tuvi
Created:	24 June 2004
Updated:	1 July 2004
Printed:	9 July 2004

This document describes the user rights management framework part of the security architecture of LATTE.

The purpose of this framework is to provide the capability of providing or restricting functionality to the user who is using the LATTE environment. It also addresses the issue of letting the scripts dictate their own security measures using the same framework.

Overview:

Concepts:

Permissions: A permission is the lowest level of restriction that can be applied to a user. If a permission is granted to a user then that means the user is allowed to use a certain feature of the LATTE system or script. It has two members: Permission name and permission description.

Roles: Roles are a collection of permissions which can be assigned to a user. This is how a user can be associated with a number of permissions. It also allows easily assigning a group of permissions to multiple users. It has two members: Role name and a list of permissions.

Users: User contains the role(s) that a user can have. Specifying roles for a user effectively describes the permission level of that user.

A user's security information is kept in two files:
passwords: This file contains the encrypted password information for the user.
security.cfg: A config file (parsable with the ConfigParser module) that contains user roles and permissions (groups in the future?). See below for the format of this file.

Both of these files should be located in a controlled directory. passwords file will have read-write access while security.cfg will be read-only to operators and read-write to administrators.

The security in LATTE will be enabled with the -S command line switch which will contain the path for these files. If this switch is not specified then security will be disabled.

In LATTE currently the following permissions are defined:

- Python shell access
- Setting the particle type
- Setting the orientation
- Enabling or disabling Global panel
    - Starting the Message Logger
    - Starting Env Monitoring
    - Starting Register Browser
    - Starting the browsing of exported files
- Enabling or disabling Power panel
    - Enabling/Disabling FrontEnd power up
    - Enabling/Disabling FrontEnd power down
- Enabling or disabling ACD panel
- Enabling or disabling CAL panel
- Enabling or disabling TKR panel
- Enabling/Disabling Preferences
    - Setting directories
        - Editing repository directory
        - Editing script directory
        - Editing data directory
        - Editing log directory
        - Editing report directory
        - Editing snapshot directory
        - Editing export directory
        - Editing runId.cfg directory
    - Setting options
        - Setting the data archiving
        - Setting the message logging
        - Setting the message logging filter
        - Setting the environmental monitor logging
        - Setting the snapshots
        - Setting the data export
        - Setting the version info collection
        - Setting the standalone elogbook integration
    - Setting run conditions
        - Setting the site run condition
        - Setting the instrument type run condition
        - Setting the phase run condition
    - Allowing User maintenance
        - Enable the deletion of a user
        - Enable user modification other than self
    - Changing system preferences
        - Setting PYTHONPATH
        - Setting elogbook url
        - Settting the data distribution server entry
        - Setting the font and style

Format of security.cfg

```
[users]
# If a user is not listed then he/she has no permissions
# The administrator role is a special role which automatically
# grants the user all the permissions.
# There may be more predefined roles which contain permissions
# based on a category.

# userid = role1, role2, role3,…
stuvi = administrator, tkr_administrator, cal_administrator, acd_administrator
claus = operator, tkr_operator, cal_operator, acd_operator
panetta = power_user

[roles]
# role_name = perm_name1, perm_name2,…

operator  = set_particle_type, set_orientation, …
power_user = allow_python_shell, …
tkr_administrator=…
tkr_operator=…
cal_administrator=…
acd_administrator=…

[permissions]
# permission_name = Permission Description
set_particle_type = Allow setting of the particle type
set_orientation = Allow setting of the orientation
allow_python_shell = Allow access to the python shell

# Subsystem script defined permissions are provided through
# the permission registration api therefore are not persistent.
```

Security Manager class (rcSecurityMan):

This class provides the necessary API for LATTE and subsystem scripts to access and modify permissions for a user and maintain the password database. It contains the following methods:

`registerPermission(role, permission, permDesc):` Method available to be called by a user script that allows the script to register a permission given a role and update any user permissions who belong in that role. Note that this requires the subsystem roles to exist beforehand. For example if the tkr_operator role is already assigned to the users then registering a permission with the tkr_operator role effectively grants that permission to the users who belong in that role.

`checkPermission(user, permission)`: Checks if the user has been granted the specified permission.

`role_has_permission(role, permission):` Check if the permission exists in role.

`getPermissions(role=None):` Returns the list of all permissions defined. If role has been specified then returns all permissions defined for that role.

getPermissionDescription(permission**):** Returns the description for the specified permission. If permission does not exist, returns None.

getRoles(self**):** Returns the list of all roles defined.

authenticateUser(loginId, password**):** Authenticate the user based on the given login id and password.

changePassword(loginId, password, userName, oldPassword=""**):** Change an existing user's password.

addPassword(loginId, password, userName**):** Add a new user to the password database.

checkPassword(loginId**):** Checks if a user already has a password.

User class (rcUser):

This class is used to store security and identification information about users. It contains the following methods:

getId(): Returns the three digit user id

getLoginId(): Returns the user's login id

getName(): Returns the user name

getRoles(): Returns the list of roles that this user belongs in

getPermissions(): Returns the list of permissions that this user has

addRole(role**):** Adds role to this user's roles

addPermission(permission**):** Adds permission to this user's permissions

isAdministrator(): Returns the user's administrator status

Notes:

The security mechanism is meant to be “operator proof” in a “kiosk” environment. It is still possible for script writers or users with sufficient privileges to change the user rights by calling the api or modifying the files manually. In addition, the user who has been granted the Python Shell permission can still alter other permissions.