Proposals on a Secure RMI Connection for Client Applications

Version 1. December 23, 2003.
A. D. Petrov, apetrov@fnal.gov

Background

In the current design of the Java controls system, all the data are provided by the Data Acquisition Engines (DAEs). In most cases, the DAEs are running on a secure subnetwork, protected by the firewall. In order to read or set the data, an application needs to access the DAE using Remote Method Invocation (RMI) protocol. The existing framework designed for the client-server connection is quite powerful and comprehensive, but does not provide enough security. Because of that, the access to DAEs is controlled mostly by the firewall which applies strong constraints on the network traffic. This leads to some serious limitations for client applications. Roughly speaking, no applications are allowed to access (read and set) the data from outside a very limited area.

In other hand, most of the client applications being designed in Accelerator Controls department are supposed to be used by a broad community of people—at least, not only by operators from the Main Control Room (MCR). The accelerator data are vital for the employees and users who may run their applications from many locations, both inside and outside the laboratory. The common approach used now consists in implementation of an additional layer between DAEs and applications. In most cases, this is a servlet which can communicate to DAE on RMI, at one side; and to a client application via HTTP, at another one. This way allows the user to start ad hoc reading jobs and get a result in some form. It does not provide much security, but protect the DAE cluster against denial-of-service attacks. Fortunately, our system does not contain any sensitive data. Yet, this approach has a number of serious disadvantages:

- it is much slower than direct RMI connection;
- it is synchronous by nature, and does not allow the client to have flexible callbacks;
- developers need to have their own formats of requests and data;
- it is hard to pass big bunches of data (e.g., for graphical applications);
- it is very hard to use both ways in one application (direct RMI for MCR and servlets for others);
- after all, it is still unsecure—hence, settings can not be allowed.

The Java controls system must offer a common way for the users to get data invariantly, from any physical location. There are at least to major solutions:

1. Secure RMI connection to DAE.
2. Developing of some common-purpose set of servlets plus some client-side API in order to communicate with them.

In either way, some secure transport level, authentication, and authorization protocols have to be implemented. However, in the second solution we still have to deal with performance issues and synchronism which are peculiar to servlets. So, an implementation of the secure RMI connection looks preferable. The following pages contain a brief review of this approach.

Task Definition

The proposed system is destined to provide a secure RMI connection between client applications and Data Acquisition Engines, in order to support reading and settings data from/to the accelerator controls system for most of the applications, and from any location. It can be considered as a part of Application Framework that supplements, but not substitute, the existing data acquisition system.

The preliminary name of the project is SCF—a Secure Controls Framework.

The proposed system should satisfy the following requirements:

1. SSL is used as a secure transport level.
2. Kerberos V5 is used for the user authentication on the client side. The system uses pre-existing (cached) valid Kerberos ticket as a user credential, but does not provide itself a way for the authentication (entering password). Upon ticket expiration, the server silently asks the client to confirm its credential. MCR workstations are authenticated implicitly without Kerberos, by IP address.
3. All security is in the server. The server authorizes the user by verifying a forwarded Kerberos ticket.
4. The client user privileges are defined by the user's Kerberos principal. Node-user pair is not used for the authorization.
5. Client-side code does not have direct access to the database or other external resources. The system provides some common API to access persistent database objects through RMI.
6. Every remote stub on the server side has its own fixed port.
7. Only trusted generic code from JDK is used for the transport level, authentication, and authorization. Critical proprietary code is readable and available for public revision.

8. The system fits (by supported data types, callbacks, etc.) at least 90% of existing applications that require access to the controls system.
9. Current way of data acquisition (job, source, disposition, item, event, etc.) stays untouched. In all other places, generic APIs provided by SDK are used whenever possible.
10. SCF client-side API is described in common terms, and independent from underlaying implementation (ACNET, VMS users, etc.).
11. The client-side part is *reasonably* small.
12. All activity is logged on the server side.

Existing Tools

In the recent time, Kerberos V5 was accepted in Fermilab as a basic protocol for user authentication and corresponding tools has been deployed on most of the servers and workstations. Though there are still some talks whether it is good or not, Kerberos is in use and do work. The current release of Java SDK 1.4 provides pretty good support of Kerberos, so called JGSS API. At present time it is used in Application Index web interface for the user authentication.

SSL/TLS, as a main secure transport protocol is also supported by Java. Moreover, the existing implementation looks even excessive for our tasks (various types of SSL authentication).

JGSS and SSL/TLS are included in SDK, and no additional jars are required.

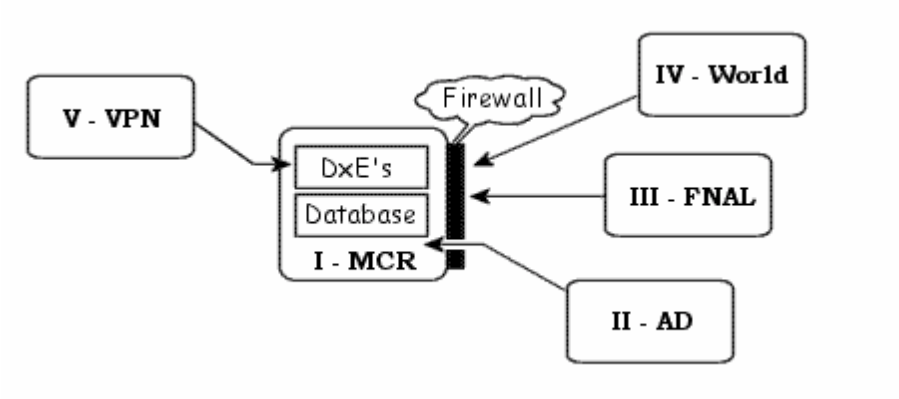
Implementation Details

Integration with existing DAE's

SCF server side is implemented as a module, pluggable in existing DAEs. Only designated engines have this module, not all. SCF module verifies the user's credential and then works as a client delegatee: translates user communications back and forth between DAE and the client in order to start a job, receive callbacks, and so forth. Database access is implemented inside server-side SCF without disturbing DAE.

Network Architecture

Currently, the entire network can be divided on 5 logical zones. Affiliation of the computer with a particular zone is defined by the computer's IP address.



I. **MCR**—Main control and computer rooms. Engines and the database are located here. Access to this zone is controlled by the firewall.

II. **AD**—Accelerator Department. Has access to Zone I through the firewall.

III. **FNAL**—the rest of the laboratory. Does not have access to Zone I.

IV. **World**—the entire world outside Fermilab. Does not have access to Zone I.

V. **VPN**—Virtual Private Network. Mapped inside Zone I.

It is proposed to deploy some number of DSE's with attached SCF modules, and allow access through the firewall to the known list of SCF's ports from specific zones. In order to protect against denial-of-service attacks, each zone should have its own group of designated server—a crash of one group will not affect the others.

The list of workstation that have implicit authentication without Kerberos is stored in the database or configuration file.

Translation of Privileges

Three kinds of privileges required by the engine to start a job are generated by SCF module on the server side.

User Privileges are defined according to the user's Kerberos principal, using existing *(?)* mapping between Kerberos principals and VMS user names. Client side of the system should not know anything about VMS user names. Kerberos principal derives from the cached ticket, verified, and the client application can not alter it. Kerberos principal of the client user is the most trusted information, because every principal is tied with a ticket. Validity of the ticket may be confirmed in the Kerberos database.

Node privileges are defined in the server side by the client IP address, using existing table of registered hosts. (*Do we need node privileges at all?)*

Service Privileges are defined according to the application's main class using Application Index registration records. Presumably, there is a chance that

application may be forged—so, service privileges in general should not be trusted *too much.*

Additional privilege attributes (default unlock, maximum unlock time, etc) are translated accordingly.

Database Access

Many* applications need to have access to some persistent objects described in the database. They are: ACNET devices, parameter pages, or other kind of composite devices, and various repositories. An ability of search and securely update this data is highly desired.

As stated above, the client may not have direct database connection, but should be able to get these data through RMI. In order to avoid creation of multiple proprietary interfaces for that, I propose to design a single scalable directory service, built in SCF and based on Java Naming and Directory Interface (JNDI). Its initial task will be an access to the device and parameter page databases. Then, additional objects may be implemented by user requests.

Conclusion

As for now, development of SCF does not look very hard. Several parts of the proposed system already work (Kerberos authentication and GSS API are used in Application Index web interface). I have conducted some research of SSL, and an implementation of secure RMI also looks feasible. My estimate is that almost everything can be implemented by the end of the spring 2004.

Java Parameter Page and Plot applications are the first candidates for the reference implementation.

However, I can not be absolutely sure that all functions proposed here are implementable. What is more important, it is unclear whether the list of security requirements is comprehensive, and the system will be allowed to have access through firewall.

* = all mine