Fermilab

Beams-doc-2495

Andrey D. Petrov
Accelerator Controls Department
Accelerator Division
630.840.6877 (phone)
630.840.3093 (fax)
apetrov@fnal.gov

October 3, 2006

Introduction to Identity Database

This technical note explains the purpose of Identity Database, possible fields of its application, and briefly describes the system's architecture and main functions.

One common problem that persists in many controls' applications is the lack of a proper access control. Many programs simply ignore the issue and rely on security constraints imposed by the environment—inability to download the code, to run the program, or to access the data. While it's probably not so bad in some cases, there is clearly a number of situations when an application shell provide the access control by itself—better granulated, with different permissions for different users, and with the log files. Yet, the authentication of client principals can be required by, or coupled with, other security mechanisms, such as transport level security (TLS). An encrypted data channel with an anonymous user guarantees privacy of that user, but doesn't make much sense to the server. In other hand, if the user is properly authenticated, it helps to secure the channel and probably to get rid of some redundancy in the protocols. Kerberos keys, for example, can be used in the TLS implementation. However, the most obvious argument for the access control is that without it the applications can eventually be denied the data access. Security policies are written by different people and get only tougher over the time.

The access control consists of two parts, authentication and authorization. In a practical perspective, the authorization task depends on a concrete application functionality. It often can be solved, though, in a way as simple as checking whether the user is a member of a particular hard-coded role. The authentication procedure is seemed to be more complicated as it requires to implement [and to configure!] one or several authentication algorithms and to have a user database. However, it can have fairly common implementation and to be self-sufficient. The need to develop and maintain the backend with a list of users and roles is perhaps the biggest hardship of this task. Even if some standard authentication mechanism is employed, there is a slim chance that the application infrastructure will be allowed to look at the corresponding user database, or that this database fits our needs.

The Identity Database (IDDB) is a set of reusable components designed to facilitate the user authentication inside a group of applications. A single instance of the system can be installed and managed in a central location, so it provides the authentication service and access to the user registration data to a certain domain of programs: standalone, web-based, or any others. The responsibility to maintain the user database can be shared (to a certain degree) between several system administrators, each of which has permissions to assign only a subset of the roles.

IDDB provides a reasonably minimal functionality to store and manage the user database, and to verify the users' credentials by the client requests. Web applications use IDDB indirectly through their containers (e.g., in Tomcat the authenticator valve is build in the application context). Standalone Java programs call the IDDB client-side API, which employs ad-hoc webservice requests to talk to the server side, so the database is not accessed directly.

The system consists of:

- The database (only 6 table, practically any RDBMS)
- Central Java code for the data access (each RDBMS requires a specific plug-in, , currently implemented are Sybase and MySQL)
- Web GUI (based on standard JSPs)
- Modules providing the interaction with particular types of applications:
  - an authenticator valve for Tomcat
  - a client-side library and API for standalone programs
  - others as needed

The IDDB server-side code talks to the database directly. It can reside inside some standard servlet—or other Java EE—container. At this time we use Tomcat 5.5.15. The code should be available (in `server/lib`) for each registered authenticator valve. For the Web GUI, the code should also be available in the web application context (as it's needed to edit the data). The web service for client applications is under development.

IDDB can support several forms of authentication simultaneously. This makes sense because people can run applications from various locations with different environments and different security policies. For example, one can be required to use Kerberos credentials when working from his office desktop, but has to enter a username and password when logging in from a public terminal in a control room, because that's not safe to obtain individual Kerberos ticket over there. In all cases, the principals this individual might have for each form of authentication are mapped to a single user account in IDDB.

Currently supported authentication protocols are:

- Username and password (passwords are stored in a hashed form)
- Kerberos (requires existing ticket)
- X.509 certificates
- IP address (for unattended processes rather than human beings)

The database stores the following information:

- Minimal information about users and roles: *id*, *name*, *info*, *email* (it's assumed that more detailed data can be looked up on the web; *email* is currently a "foreign key")
- One or several authentication principals for each user
- Role assignments to users and other roles (so, nested roles are permitted); conditional assignment is supported (e.g., *"user* gysin *has role* wheel *only when she's authenticated through a password"*)
- Access log

In a perspective, IDDB will support federation of identities between several instances of the system, through some standard protocol, such as SAML.