USER AUTHENTICATION FOR ROLE-BASED ACCESS CONTROL

Andrey D. Petrov, Suzanne Gysin, and Carl Schumann,
Fermi National Accelerator Laboratory, Batavia, IL 60510, U.S.A.

Abstract

The user authentication system is a part of the Role-Based Access Control (RBAC) project for the high-level LHC Control Software (LSA) at CERN. The project is being developed by LAFS ("LHC at Fermilab Software") collaboration between control groups at CERN and Fermilab. The function of RBAC authentication is to create, distribute, and manage digital credentials for the users. We had to consider many constraints dictated by the existing control system, and diversity of the used software. This paper describes the general design and implementation of the authentication system in Java and C++. We also give and overview of its additional features, such as Single Sign-On and Role Picker.

PURPOSE OF THE PROJECT

As described in Role-Based Access Control overview [1], we separate two principal concepts in the design of RBAC: authentication (A1) and authorization (A2). Both parts are implemented independently, as two different systems, which do not interact in any way other then passing the users' credentials from A1 to A2.

As appears from its name, the purpose of the RBAC authentication system is to verify the digital identity of a principal (which is either a human user or a program). This can be accomplished in several ways, described below. In any case, if the authentication succeeds its result is a digitally signed authentication token that is returned to the application. The program can use the token whenever it needs to interact with various parts to the control system. For example, the token can be provided as one of the arguments in an RMI call to set a device. Front-ends and the middleware that are receiving such calls will verify the token, thus confirming the identity of the remote party, and can use it as a base for authorization.

The RBAC authentication token is a short-term uniform substitute of the real credentials. It gets issued by a central service that can reliably verify the user's identity. Various recipients of the tokens can validate them quickly and easily, and use for making authorization decisions.

Because the RBAC project began when all other parts of LHC Control Software have been already completed, its design was the subject to a number of requirements and limitations dictated by the infrastructure in place. Basically, we couldn't change much about how the system operated, so RBAC was built as an additional part on top of the existing components.

The RBAC authentication system was not designed to be absolutely secure, or to sustain any possible kind of attacks. It's supposed to be used in a certain environment, were other means of protection, such as network firewalls and monitors, as well as the physical access control, are in place. Based on the analysis of real threats, RBAC was built to protect mainly against human errors, rather than against deliberate efforts to break the system down. The overall design of the authentication system was the result of a trade-off between performance, security, and complexity.

TECHNICAL DESIGN

The general layout of the RBAC authentication system is shown on Fig. 1. It's built on a common client/server model.

The A1 server receives authentication requests via HTTP from multiple clients, returning back either an authentication token, or an error code. Each request from a user contains its credential in some form. All requests are atomic, so no session information is cached by the server. The SSL/TLS protocol is generally used over HTTP to protect the communication between the two parties, and to authenticate the client's X.509 certificate, if such is provided.

The client side is organized as a library, which can be used by other applications or application frameworks. This library provides a function that should be called in order to obtain the authentication token from the server. It also provides several standard GUI components, such as a login dialog, role picker dialog, and others. Basically, that authentication client-side library can be used in most application without changes. Yet, its behavior can be customized depending on a particular use case. For example, for a headless program the user name and

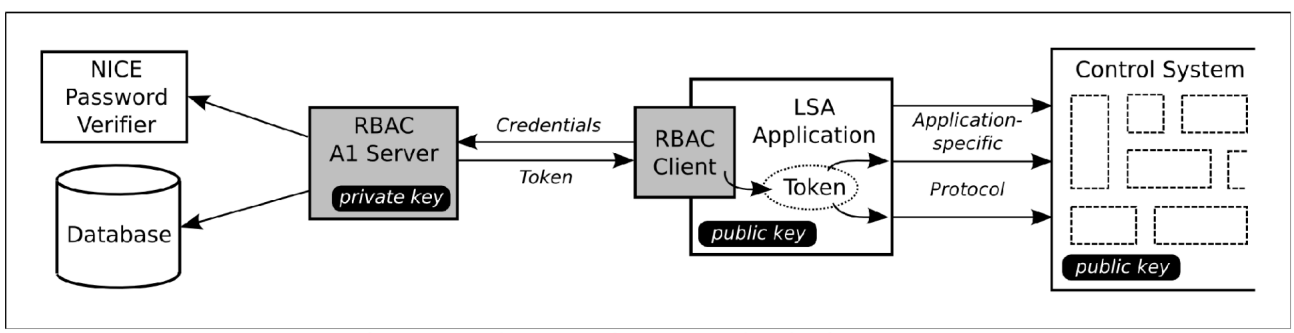

Fig 1: General Layout of RBAC Authentication System

password can be obtained by a way other then popping up a graphical dialog.

There are also two back-end services required by A1 server. One is used to verify site-wide user names and passwords. Another one is an Oracle database. Both of them are parts of the central IT infrastructure at CERN, and do not belong solely to RBAC. These services are contacted only by A1 server, and never directly from the clients.

TYPES OF AUTHENTICATION

RBAC supports four types of authentication:

- By the user name and password.
- By a X.509 certificate.
- By the network address.
- By using an existing authentication token.

Password Authentication

The user names and passwords are checked against the central NICE [2] account database, via a dedicated web service. No user account information is stored in the RBAC own database.

Certificate Authentication

If the user's X.509 certificate is available, it can be applied in the standard client authentication mechanism of TLS/SSL protocol. Then, the certificate information is used to look up the user name in the RBAC database.

Address Authentication

Certain clients can be authenticated by their IP addresses, using a lookup table in the RBAC database. Normally, the address authentication is permitted only for a very limited number of machines, such as control room consoles.

Authentication with Existing Token

Any existing token can be used to request a new one, providing that the original token is not expired, bears valid signature, and was issued to the same location address. The validity time of the new token will not exceed the validity time of the original one.

This function of “re-authentication” is utilized for several purposes, such as Single-Sign On and Role Picker, described below.

AUTHENTICATION TOKEN

The authentication token is a chunk of data that proves the digital identity of its possessor for a limited period of time. It is created on A1 server and gets digitally signed with the server's private key.

Once the token is created, the server sends it back to the client, which, in its turn, sends it to other parties when it needs to identify itself. The client program can reuse a single token many times, unless it expires. The RBAC design document [3] standardizes the binary format of the token, so that it can be implemented in many programming languages and on various platforms. Although the token incarnates to different kinds of objects (or structures) in different programming languages, it is guaranteed that multiple serialization and de-serialization do not change its binary form and do not break the signature.

Each token holds three groups of the data fields: (1) authentication data, (2) auxiliary data, and (3) a digital signature.

The first group, the authentication data, is intended for regular recipients, using the tokens for authorization. It includes the following fields:

- Serial Number (a random integer)
- Authentication Time
- Expiration Time
- Application Descriptor
- Application Timeout
- Location Address
- User Name
- List of Roles

The second group, an auxiliary data, is used solely by the authentication system itself, and should be disregarded during authorization. It may include a variety of fields, such as a full list of roles for the user, a type of the token, etc.

The third group contains a digital signature calculated over the first two groups. None of the data in the token is encrypted.

VERIFICATION OF THE TOKEN

Before the token is accepted by the authorization part, it must be validated. The client applications may also want to validate tokens occasionally.

The token is considered valid if its digital signature can be verified using the server's public key, and its expiration time is greater then the current time. Because the validation procedure is so simple, it can be conducted very quickly inside the middleware and front-ends, without contacting the server. Furthermore, RBAC does not offer any central validation services (such as revocation lists).

DIGITAL KEYS

The RBAC authentication system requires only one principal keypair (a matching combination of private and public keys for asymmetric cryptography). This set of keys identifies the authentication server before all the clients. The private key is known only to A1 server, where it's deployed appropriately. The public key is made available for all applications and front-ends. Normally, it gets distributed along with the RBAC client-side code (for Java applications, inside the program's .jar file).

The keys are employed for two purposes. Firstly, the server uses its private key to sign all outgoing tokens, so that the recipients can verify their authenticity. Secondly, the server's private key is used to identify it before the clients via SSL/TLS protocol. As the matching public key is known to everyone, the clients can check whether the server is trusted before sending passwords to it.

ADDITIONAL FEATURES

Besides the obvious form of authentication, when the token is generated in reply to explicitly provided credentials (such as user name and password), RBAC supports two additional functions, based on re-authentication of an existing token.

Single Sign-On (SSO) allows the user to log-in only once, and then reuse the credentials in several applications running on the same machine. As every token is application-specific, it can't be reused by multiple programs directly. Instead, when SSO is activated, the RBAC client creates a "master" token that doesn't belong to any particular application and is not good for a normal use. Yet, every application can read it from a cache and use it to request its own token.

The Role Picker can be used to reduce the number of roles the current user possess. For example, when performing some operation, a system administrator may want to get rid of some privileges. Instead of editing his profile in the user database, one can activate the Role Picker dialog and de-select the roles that are not desirable.

CURRENT STATUS

Currently, the RBAC authentication system is released in a production version.

The A1 server is implemented as a set of servlets running on Jetty 6. To improve performance and reliability, several instances of the server are running. The clients connect to them in a random order.

The clients-side library is implemented in Java and in C++. Not fully supported feature at this time is X.509 authentication.

REFERENCES

[1] S. Gysin, A. D. Petrov, et al., "Role-Based Access Control for the Accelerator Control System at CERN"—TPPA04, ICALEPCS'07, Knoxville, October 2007, http://neutrons.ornl.gov/conf/icalepcs07/

[2] NICE Services, https://websvc06.cern.ch/winservices/

[3] S. Gysin, K. Kostro, et al., "Role Based Access for the Accelerator Control System in the LHC ERA"—Requirements, LHC Project Document No. LHC-C-ES-0007.