ROLE-BASED ACCESS CONTROL FOR THE ACCELERATOR CONTROL SYSTEM AT CERN

S. Gysin, A.D. Petrov, FNAL, Batavia, IL 60510, U.S.A.
P. Charrue, W. Gajewski, V. Kain, K. Kostro, G. Kruk, S. Page, M. Peryt
CERN, Geneva, Switzerland

Abstract

Given the significant dangers of LHC operations, access control to the accelerator controls system is required. This paper describes the requirements, design, and implementation of Role-Based Access Control (RBAC) for the LHC & injectors controls systems. It is an overview of the two main components of RBAC: authentication (also called A1) and authorization (A2), and the tools needed to manage access control data. We begin by stating the main requirements of RBAC and then describe the architecture and its implementation. RBAC is developed by LAFS, a collaboration between CERN and Fermilab.

The technology used for authentication and authorization are discussed in separate papers also in these proceedings [1][2].

MOTIVATION

The main motivation to have RBAC in a control system is to prevent unauthorized access. RBAC is a preventative and therefore inexpensive way to protect the accelerator. RBAC keeps a user from making the wrong settings or from logging into the application in the first place. Other machine protection systems such as interlocks are reactive and once triggered it is expensive to recover operations. Prevention is much less intrusive.

RBAC is also used to ensure machine stability during a run. Once the machine is fine tuned and beam is in the machine, an errand setting can disrupt operations for hours and loose valuable data. With RBAC, one can disallow settings during certain machine states, and hence ensure stability.

The third motivation is that each setting protected by RBAC is logged. This is critical during commissioning and debugging. Each setting can be traced and bugs in the sequencer or operations can be caught and corrected.

REQUIREMENTS

The requirements for RBAC for the LHC controls were well defined and the scope was well contained. We authored an explicit requirements document [3] in EDMS and went through a formal approval process. Basically, RBAC authenticates a user, assigns roles to users, and protects device properties with permission settings for roles. Because the scope was so well defined and contained, we were able to write and deploy RBAC very quickly.

IMPLEMENTATION

RBAC works by giving people roles and assigning the roles permissions to make settings.

Roles

A role is usually a job function, such as LHC-EIC (Engineer in Charge), LHC-Operator, or PO-Calibrator (Power converter calibrator). A user can have multiple roles at any time, and any given user can be a member of any given roles

Roles are created by a Role-Maker, a person who has the Role-Maker role, which is just another role in the RBAC role list. The request for a new role specifies the role name and identifies an administrator. The administrator has the responsibility to manage the membership of the role. A user receives his roles when logging in.

A screen capture of the RBAC browser [4] showing roles and users is shown in figure 1.

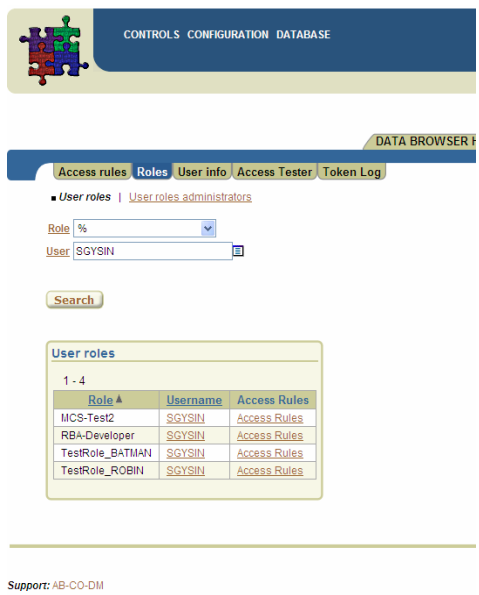

Figure 1: RBAC browser test roles and users.

Access Rules

The permissions are set on CMW device properties which map to devices defined in the device database (power converters, collimators, kickers, etc.) The types of access that can be granted or denied are: set, get, and monitor.

The device properties are protected with access rules. These rules specify what roles can make settings. Besides

the roles, there are 3 other parameters that can be specified in an access rule: the location (i.p. host address), the application, and the accelerator mode.

An access rule is created by the access rule administrator. Each device class has an access rule administrator. Similar to the role administrator, a rule administrator manages the rules for the specific device class. A Rule-Maker creates a new device class and its administrator.

RBAC Tokens and Access Maps

RBAC provides several ways of authenticating a user. The most common one is NICE, the authentication the integrated environment for all the PC users at CERN. Specifically, a login dialog is presented and the user types in the username and password. As a result of successful authentication, RBAC sends the application an RBAC token. This token contains the user name, the roles, the application name, the location, the time and date of issue, an expiration time, and a digital signature.

The digital signature is created by the RBAC server with the private key, and can be validated with the RBAC public key. This insures the integrity of the token, since the signature contains the digest of the content, and it verifies that the token came from the RBAC server since only it has the private key. It is important to mention that RBAC is not a security system against hackers; it is designed only to prevent well meaning people from making the wrong setting, and people who have no credentials from running a controls application.

On the front-end, an access map is created containing only the relevant access rules. It is loaded in the front end memory.

The token is passed by the application to the Controls Middleware (CMW) with the request to make an access from the application. The CMW client passes the token to the CMW server validates the signature and the expiration time of the token. If the token validation is successful, the CMW server checks the access maps for permission. If that check is also successful the request is sent to the equipment [3].

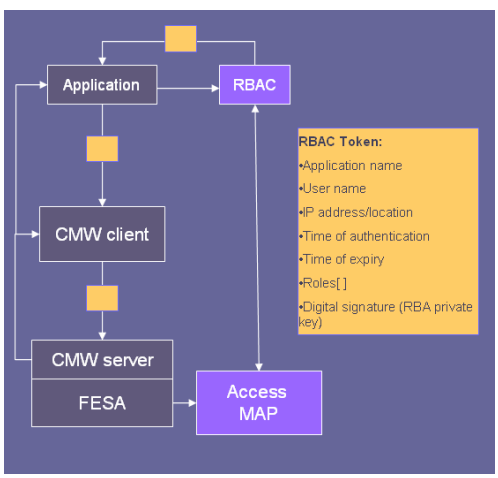

Figure 2: RBAC tokens and access maps.

The basic RBAC design is that the user receives an RBAC token with the roles, and the token gets sent to the CMW server where the access rules are checked. According to the check, the users request is denied or accepted and sent to the equipment. The equipment only gets authorized requests.

The default, at this time, is when there is no rule the property is unprotected. Once there is a single rule on the property it is protected against everyone else not explicitly mentioned in the rule.

Special Features

In addition to this basic functionality there are many special features.

- Authentication by Location: an access rule can limit the access on a location basis. Locations are set in the database on a host address basis.
- Authentication via X.509 certificates.
- Authorization by application: an access rule can limit the access for applications.
- Role Picker: a user may pick a specific role if multiple roles are available
- Generate and manage public and private keys for Critical Settings
- Token Log (currently over 140,000), keep track of who receives tokens and if they were rejected.
- Load balancing (2 RBAC servers)
- Web interface for browsing and editing roles and rules.[4][5]
- Access map builder by class, server, and front end.
- Application timeout: authentication timeout for an application.
- Role timeout: authentication timeout for a specific role (temporary role)
- Single Sign-on: once logged into one application, the others pick up the token.[1]
- Temporary (EIC) roles

DEPLOYMENT

The RBAC server is a Java web application running in a Jetty container. It is deployed as a war file and installed on two production, and one test Linux servers on the CERN technical network. It accesses CERN's production database and uses the SOAP web services provided by NICE [8].

The RBAC client, which is the login dialog and other classes used by the application, are deployed in a jar file.

Also available is a C++ authentication library for C++ clients to acquire a token, for example LabView.

The RBAC component in the CMW that validates the token and checks the access maps is written in C++ and available as a stand alone C++ library.

The token format of C++ and Java has to match exactly because it is created in Java and validated in C++. We have implemented the RBAC token in CORBA IDL. The C++ and Java classes are generated automatically at compile time.

There is also a Users Manual available [6].

PERFORMANCE

An important concern with any design is its performance. If RBAC would double the time to make a setting, nobody would use it. The system also had to scale with a large number of access rules. How does a 2000-rule access map on a front-end impacts the search for the right permission? Another potential performance hit was logging each request on a protected property, and the 3-tier vs. 2-tier performance.

LHC controls can run in 2-tier mode, meaning the application and the client are on one machine and the database and devices are accessed directly on the second machine. This configuration is used when in developer testing. However when the system is in operation the configuration is usually 3-tier. Meaning the client is on a middle tier. A common client is shared by several applications and consolidates requests. In 2-tier, a dedicated client ensures that each request is made from the same application. The RBAC token can be validated once per session and the credentials can be used for each subsequent request.

In 3-tier, the requests can come from any application at random times. The simplest design is to verify the token for each request. But how does this affect performance?

We ran performance tests to answer these questions on a front-end with a 400 MHz. Power PC with Linux, and here are the results:

- The size of the access map has very little effect on performance due to sophisticated and optimized search algorithms. Our test result show a 0.02 ms increase between a 20 rule and 2000 rule access map.
- Logging each request also has very little effect on the performance. Our tests show the difference between having logging on vs. having it off is 0.003 ms.
- 3-tier token verification on every request has a larger impact on performance than the other two concerns. The key size is the most contributing factor. A DSA, 1024 bit key takes 5 ms to validate. A RSA 512 bit key takes 0.150 ms. The smaller key is an order of magnitude faster. For a 2000 rule access map in a 2-tier configuration the average turn around time or a request is 0.7 ms. In a 3-tier configuration it is 2.7 ms. At this time, this is acceptable according to the requirements, and if we deem this too slow we can make improvements.

These performance results are mentioned here because anyone starting to design an RBAC system will have performance concerns. This turns out not to be the most difficult challenges. The most difficult challenge of RBAC is not technical at all, but procedural or social. It is who decides what roles should be created and what access they should have.

STATUS

RBAC was developed by LAFS and CERN's Accelerator/Beams Control and Operations groups. LAFS is a team at Fermi National Accelerator Laboratory that collaborates with CERN to develop controls applications.

RBAC was tested, and released for production in June'07. Currently there are about 50 users, 25 roles, 10,000 rules, and over 140,000 tokens have been requested.

REFERENCES

[1] Andrey Petrov, Schumann Carl, Suzanne Gysin (Fermilab, Batavia, Illinois), "TPPA12 – User Authentication for Role-Based Access Control", ICALEPCS'07 proceedings.

[2] Krzysztof Kostro, Wojciech Gajewski (CERN, Geneva), Suzanne Gysin (Fermilab, Batavia, Illinois), "WPPB08 – Role-Based Authorization in Equipment Access at CERN". ICALEPCS'07 Proceedings.

[3] S. Gysin (Fermilab, Batavia, Illinois), K. Kostro AB/CO, G. Kruk AB/CO, M. Lamont AB/OP, S. Lueders IT/CO, W. Sliwinski AB/CO, P. Charrue AB/CO. "Role-Based Access for the Accelerator Control System in the LHC Area – Requirements" http://wikis.cern.ch/download/attachments/7078162/LHC-C-ES-0007-10-00.pdf?version=1 .

[4] M. Peryt, CERN: RBAC Data Browser: http://oraweb.cern.ch/pls/htmldb_dbabco/f?p=CONFIG_BROWSER .

[5] M. Peryt, CERN: RBAC Data Editor: https://cs-ccr-oas4.cern.ch/cce/faces/rba/RBAHome.jspx .

[6] S.Gysin, "RBAC User's Manual" http://wikis.cern.ch/download/attachments/7078162/Users+Guide.doc?version=7 .

[7] RBAC Wiki: http://wikis.cern.ch/display/LAFS/Role-Based+Access+Control .

[8] NICE SOAP web services: https://espace.cern.ch/authentication/CERN%20Authentication%20Help/SOAP%20WebServices.aspx .