LHCC Poster Session – CERN, Feb. 2018

The ATLAS Access Manager Policy Browser

Introduction

The ATLAS experiment comprises a significant number of hardware and software resources accessed and operated by many users. The Access Manager implements the resources access restriction to reasonable levels required for the successful running of the ATLAS experiment. The total number of the ATLAS users from CERN and external institutes is quite significant, so the Access Manager only grants permissions to users following their current duties.

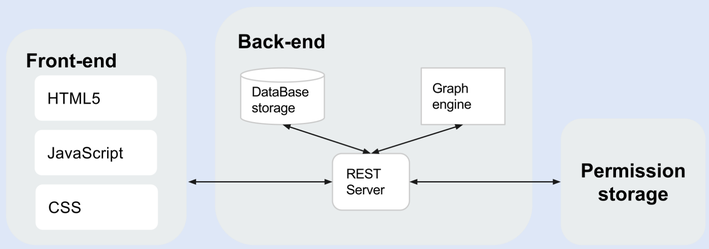
The service is designed on top of the Role Based Access Control (RBAC) [1] model. Every role describes a set of actions granted to users. The actions allow login to a set of machines, execute some commands with the security privileges of another user (sudo), or perform TDAQ specific operations. The roles can be defined using inheritance. Every ATLAS user has a well-defined set of access privileges corresponding to a specific set of assigned roles. In total, there are several hundred roles and several thousand users.

Given the size and the complexity of the system, a tool to browse and inspect the Access Manager configuration is required. We present the deployment of a visualization tool named Policy Browser. Currently, it is the primary tool for role administrators to inspect all the aspects of the Access Management via a rich web-based interface.

Role Based Access Control (RBAC)

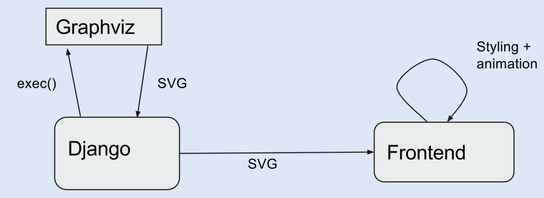
The RBAC model takes the access decision for an individual user based on the roles the user has in the organization. A role gives access to one or more permissions. A permission is the right to perform an action on a resource.

Back-end Overview



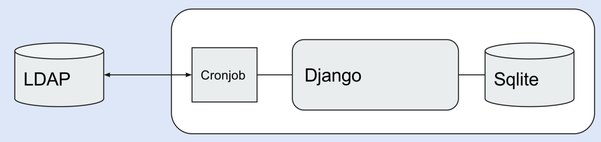
The Policy Browser is a web application exposing a REST[2] API. The server produces data (JSON) and graphs (SVG) and sends them to the client that displays them. The server is using Django[3] a Python framework that follows the model-view controller (MVC)[4] architecture. The selection of Python was particularly interesting to do short prototyping iterations at the beginning of the project and the MVC architecture allowed to focus code development on the specific features of the project.
The client is using AngularJS[5] an open-source JavaScript framework in conjunction with Bootstrap[6]. It allowed producing a state-of-the-art GUI in a short time span.

Back-end Graph Engine



The roles are organized into an inheritance hierarchy reflecting different levels of users expertise. Often for a given, role it is necessary to know what roles it inherits from and which ones are derived. There are various ways to draw graphs. Comparing them objectively requires defining appropriate metrics. In the case of a dependency graph (directed acyclic), a common approach is the layered graph drawing (also called Sugiyama drawing). Ideally no edge should go upward, and the goal is to minimize the number of edge crossing for readability. As for a large graph this can be a costly operation, the server relies on a dedicated graph engine, Graphviz[8], for all the graphs generation.

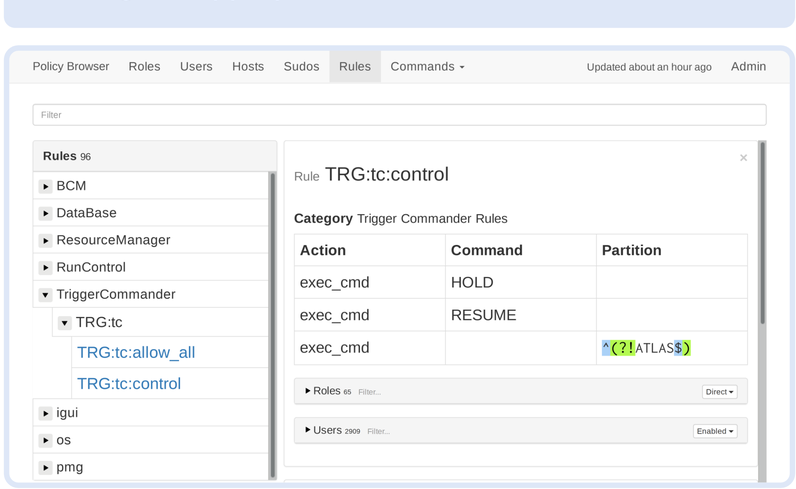
Back-end Storage & Data Update



The actual permissions used by the system are stored on LDAP server.
Some requests require complex manipulations with its data. The first in-memory prototype resulted in a complicated and hardly maintainable codebase.
The second implementation leveraged usage of a relational database (Sqlite[7]). It allowed to make the code more readable and to benefit from the query optimizer for a fast response time.
Django ships with an Object-relational Mapping (ORM) that accelerates the initial design of the schema. A process periodically checks if the permissions have changed, and updates the database accordingly.

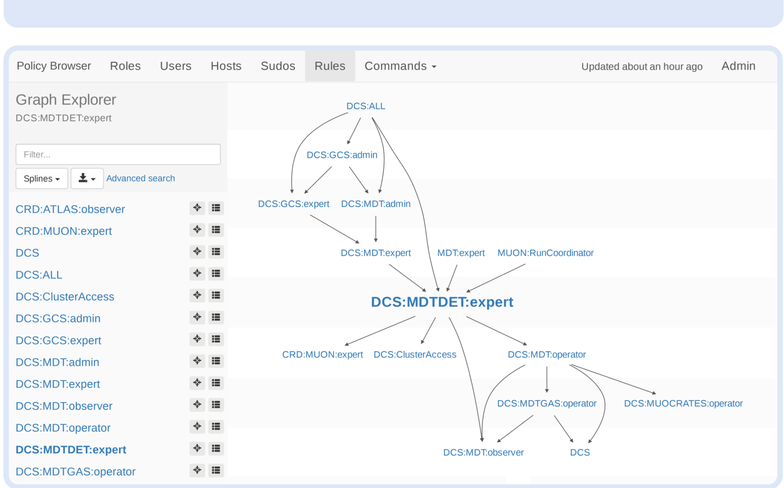
Front-end Overview

The web interface consists essentially of data panels to display lists of roles, users and the various permissions. A tree display based on resource name allows a compact representation and prevents having to browse list comprising thousands of elements. For a specific object, we display the list of related authorization (e.g. for a given command it displays a list of roles that allows its execution and the list of authorized users). It also proposes text and category filters to further help finding the appropriate resource.



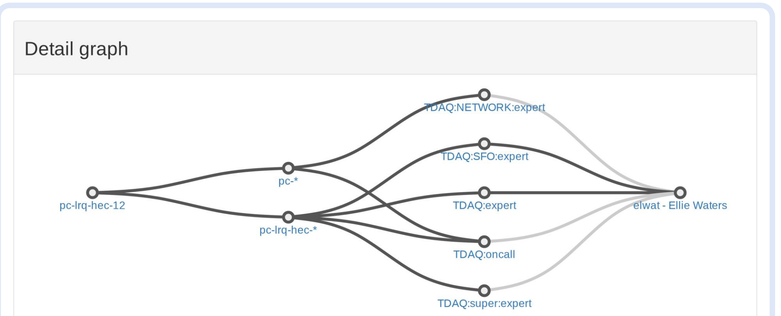
Front-end Graph Explorer

The Graph Explorer is a tool that allows displaying the inheritance hierarchy.
It allows browsing large graphs comprising many layers and hundreds of roles with zoom and pan. It permits easy navigation from role to role with a graph morphing animation.
Lateral filters are shown in order to highlight subgraph and an advanced search mode allows to fully customize the display.



Front-end Detail Graph

The role is the key element granting permissions. A user may have permission granted by several roles. Sometimes it is required not only to know which resource a user can access but via which role. This is were the detail graph is useful. Since often several paths allow access to a resource, the detail graph is an effective way to represent this information.



Conclusion

The application has been deployed to the ATLAS experiment site for production. It uses live data from the experiment. All the initial requirements have been implemented. On a performance point-of-view, the critical part of the API is the graph generation. Benchmark results show that the current implementation requires an average of 70ms per graph (CERN CentOS 7, i7-3770 CPU) which should not be perceptible by end users. The preliminary results from informal user feedback are generally very positive. They tend to indicate an intuitive layout and a general ease of use.

References

1 RBAC https://csrc.nist.gov/Projects/Role-Based-Access-Control
2 REST http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
3 Django https://www.djangoproject.com/
4 MVC https://dl.acm.org/citation.cfm?id=50757.50759
5 AngularJS https://angularjs.org/
6 Bootstrap https://getbootstrap.com/
7 Sqlite https://www.sqlite.org
8 Graphviz https://www.graphviz.org

Sami Perrin[1], Giuseppe Avolio (supervisor) [2], Igor Soloviev (supervisor) [3]
([1] Ecole polytechnique Fédérale de Lausanne, [2] CERN, [3] University of California Irvine)