Attribute-Based Access Control for NoSQL Databases

Eeshan Gupta
Indian Institute of Technology
Kharagpur, India
eeshan9815@iitkgp.ac.in

Shamik Sural
Indian Institute of Technology
Kharagpur, India
shamik@cse.iitkgp.ac.in

Jaideep Vaidya
Rutgers University
Newark, USA
jsvaidya@business.rutgers.edu

Vijayalakshmi Atluri
Rutgers University
Newark, USA
atluri@rutgers.edu

ABSTRACT

NoSQL databases are gaining popularity in recent times for their ability to manage high volumes of unstructured data efficiently. This necessitates such databases to have strict data security mechanisms. Attribute-Based Access Control (ABAC) has been widely appreciated for its high flexibility and dynamic nature. We present an approach for integrating ABAC into NoSQL databases, specifically MongoDB, that typically only support Role-Based Access Control (RBAC). We also discuss an implementation and performance results for ABAC in MongoDB, while emphasizing that it can be extended to other NoSQL databases as well.

CCS CONCEPTS

• **Security and privacy** → **Access control**; • **Information systems** → *Key-value stores.*

ACM Reference Format:
Eeshan Gupta, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. 2021. Attribute-Based Access Control for NoSQL Databases. In *Proceedings of the Eleventh ACM Conference on Data and Application Security and Privacy (CODASPY '21), April 26–28, 2021, Virtual Event, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3422337.3450323

1 INTRODUCTION

NoSQL databases have the advantage when it comes to managing high volumes of heterogeneous and unstructured data efficiently, which is ideal for a large number of applications in use today, especially web applications and Internet of Things applications that need significant scalability. Such systems, however, lag behind relational databases when it comes to data security [4]. A major security concern is that of effective access control. On the other hand, Attribute-Based Access Control (ABAC) has started gaining popularity for a wide range of applications due to its high flexibility and customized data protection ability at various levels of granularity [3]. This work focuses on addressing the challenge of integrating ABAC into NoSQL databases. As NoSQL databases are built on a diverse set of underlying data models, we in particular focus on MongoDB since it is one of the most popular NoSQL databases according to the DB-Engine Rankings [5]. We also discuss an implementation of enhancing MongoDB's underlying Role-Based Access Control (RBAC) system with attributes and dynamic assignments. The system can be integrated easily into any MongoDB server with no additional query overhead.

CODASPY '21, April 26–28, 2021, Virtual Event, USA

ACM ISBN 978-1-4503-8143-7/21/04.
https://doi.org/10.1145/3422337.3450323

2 PRELIMINARIES

In this section, we discuss the current state of access control in NoSQL databases and how it can be improved using ABAC. We also briefly describe some of the basic concepts of MongoDB.

2.1 Access Control in NoSQL Databases

Access control mechanisms form an essential tool for information security and protection by mediating which users are allowed to access which components of the organizational resources. The most powerful form of access control currently available in NoSQL Databases such as MongoDB is Role-Based Access Control (RBAC) [6]. It allows system administrators to create roles which broadly correspond to the various position in the organization hierarchy, and assign users to those roles.

However, RBAC systems are static in nature, and they depend heavily on system administrators, as well as the identity of users, as opposed to their attributes, to be able to map them to a certain set of roles. Hence, these systems are inherently less powerful, there being no support for granting of privileges on the basis of user, object and environmental attributes.

2.2 Attribute-Based Access Control (ABAC)

ABAC allows granting of permissions and privileges to perform operations on objects on the basis of assigned attributes of the user and object, and on the basis of environmental conditions such as time of day, day of the week, location of user, etc. A set of policy rules specified using these attributes and conditions is responsible for determining any access decision. Attributes are name-value pairs describing various states and characteristics of the components [7].

ABAC can provide dynamic, fine-grained and identity-agnostic access control, and is considered to be the next generation of access control systems. With no NoSQL database currently providing support for ABAC and hence lacking in data security measures, they can benefit from such systems.

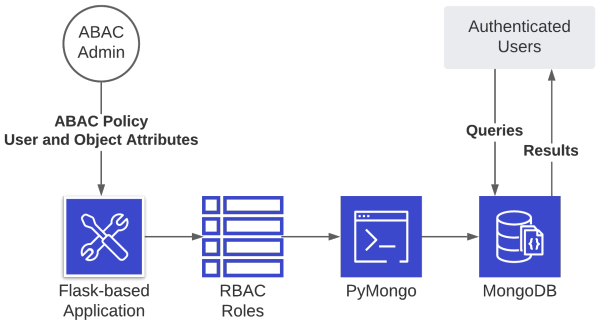


Figure 1: Summary of the ABAC System

2.3 MongoDB

As opposed to relational databases, which support well-defined Data Definition and Data Modeling languages, NoSQL databases such as MongoDB operate with a variety of data models and languages. There is no single language such as the Structured Query Language (SQL) for relational databases which works for NoSQL databases. Each vendor instead mandates a specific way for its NoSQL database to be used. This makes a general approach to integrate ABAC into NoSQL databases quite impractical. Hence, we focus on MongoDB, while emphasizing that due to its modular nature, this method is extendable to other NoSQL databases as well through simple appropriate product specific changes.

3 ENHANCING ACCESS CONTROL IN MONGODB USING ABAC

In this section, we discuss the design and implementation of ABAC for MongoDB databases. A summary of the system is given in Figure 1. The ABAC system administrator provides the user and object attributes, and the ABAC policy, which is described in terms of the attribute values necessary for a particular operation, to the Flask-based web application. These are then converted into a compact set of roles as described below. The roles are then created in MongoDB using PyMongo, after which the authenticated users can communicate with the MongoDB database according to their permissions.

The PyMongo library allows interaction with a MongoDB database through Python. This allows implementation of the algorithm mentioned in [1] using Python, and through the interface provided by PyMongo, one can communicate with the database for creating new roles based on the mapping produced by the algorithm. Hence, it is possible to leverage MongoDB's RBAC system, to develop a working implementation of ABAC.

The system works with all MongoDB drivers, servers, clients and versions, and provides easy integration in a MongoDB deployment. More precisely, it interacts with the MongoDB server with basic PyMongo commands which require administrative access to the database. Since PyMongo is the recommended way of interacting with MongoDB from Python, its compatibility with all versions of MongoDB is provided by the developers of MongoDB.

3.1 Generation of RBAC Roles From ABAC Policies

This module follows the algorithm described by Batra et al. [1]. A Python class `ABAC` accepts as input the User Attribute relation, the Object Attribute relation and the ABAC Policy. The User Attribute relation is encoded as a binary matrix, with a row for each user, and each column represents an attribute value taken by that user. Similarly, the Object Attribute relation is also encoded as a binary matrix, with a row for each object. The ABAC policy is a tuple of the User Attributes and Object Attributes required for providing a list of permissions to the user for a particular object.

These are next processed for generating the set of authorizations using the algorithm described in [1]. The time complexity of this algorithm is $O(num\_users \times num\_objects \times num\_rules \times num\_user\_attributes \times num\_object\_attributes)$.

The set of permissions for each user (User Permission Assignment) is then derived from the set of authorizations using the algorithm introduced in [1]. The time complexity of this algorithm is $O(num\_users \times num\_objects)$.

The Permission and User Assignment relations are then derived from the UPA by performing role mining using the approximation algorithm suggested by Ene et al. [2]. It generates a compact set of roles which are disjoint in their permissions. This is done by finding the minimum biclique cover in the bipartite graph from users to their permissions by fast graph reductions.

There are also functions to modify and update the set of assignments, User Permission Assignment relations, and corresponding User and Permission Assignment relations whenever a new user, a new role or a new attribute is added or deleted.

The output of this module can be interpreted as a set of roles which the given ABAC policy can be broken down into.

3.2 Generating and Assigning Roles in MongoDB

The Python program connects to the MongoDB server using PyMongo with an administrative Role to be able to create and assign roles in the server.

For each role in the Permission Assignment relation, a role is created in the MongoDB server using the `createRole` command of MongoDB, and users are assigned to each role according to the User Assignment relation using the `grantRolesToUser` command. Whenever any modifications are made to the ABAC Policy, or a new user or object is added, these methods are invoked again according to the new User and Permission Assignment relations. A system enforcing ABAC was implemented using the techniques described in the previous section and a web application was developed for the same using the Flask framework. The web application provides the functionality to add, update and delete users, objects and their corresponding attributes. System administrators can also add, update and delete rules for enforcing the ABAC policy.

Initially, before adding any rules into the ABAC Admin system, attempting to execute the Operation `find` leads to an `Unauthorized Access Error` as demonstrated in Figure 2(a). However, after adding the rules, the users get the appropriate accesses according to the ABAC Policy. Whereas the user was denied access before

adding the rules, when the `find` command was run again, access is provided as demonstrated in Figure 2(b).

Thus, this is a fully functional ABAC system for MongoDB. In addition to that, the main ideas for the integration of ABAC into MongoDB can be easily extended and adapted to other document-oriented NoSQL databases with an implementation of RBAC.

```
> db.inventory.find( {item: "canvas"}, {_id: 0} )
Error: error: {
        "ok" : 0,
        "errmsg" : "not authorized on test to execute command { find:
\"inventory\", filter: { item: \"canvas\" }, projection: { _id: 0.0 },
 lsid: { id: UUID(\"1a4aca0a-fdb0-4ad7-a2aa-d911b6628ba8\") }, $db: \"
test\" }",
        "code" : 13,
        "codeName" : "Unauthorized"
}
```

(a)

```
> db.inventory.find( {item: "canvas"}, {_id: 0} )
{ "item" : "canvas", "qty" : 100, "tags" : [ "cotton" ] }
```

(b)

Figure 2: Find operation (a) before, (b) after adding the rules

3.3 Results

This section demonstrates the performance results of the web application for providing controlled access to the users.

We present the time taken for the system to process all its inputs, and enforce the ABAC policy in the MongoDB database. The system was implemented in Python3 on a 2.2 GHz Intel i7 machine having 16 GB of RAM. Once this is executed, there is no additional overhead for the users to communicate with the MongoDB database. In figure 3(a), the number of objects, object attributes, and ABAC rules is fixed, and the variation in execution time is measured with the number of users (|U|) and user attributes (|UA|), whereas in figure 3(b), the variation is measured with the number of users (|U|) and rules (|R|).

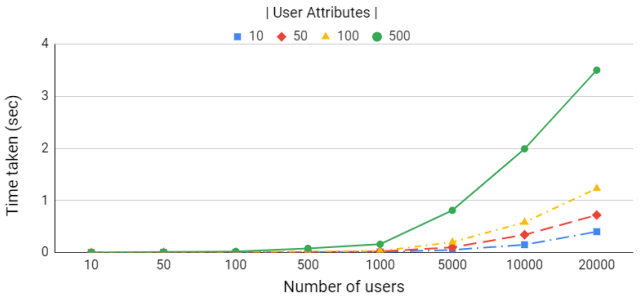
It is observed that the proposed system is efficient and hence, can be meaningfully deployed in real-life NoSQL database applications,

4 CONCLUSIONS AND FUTURE DIRECTIONS

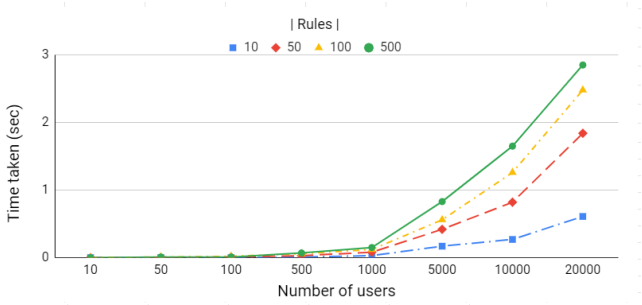
In this paper, we discussed a method for incorporating ABAC in NoSQL databases, and provided an implementation of the same for MongoDB. The implementation is modular in nature, and can be easily adapted to any NoSQL database having support for RBAC.

While the method for deploying ABAC policies using RBAC roles covers a variety of features of ABAC, there are still a few aspects in which it is lacking. This includes the full power of ABAC such as incorporating environmental attributes and dynamic assignment of users without prior registration.

To have a native implementation of ABAC in MongoDB, it would be necessary to intercept each message sent to the MongoDB server and have a proxy server that connects to the server on behalf of the client. The interaction between the client and the server in MongoDB is based on the MongoDB Wire Protocol [6]. The requests and responses are converted by the MongoDB drivers to MongoDB Wire Messages. So, intercepting each message, we can build a layer of access control in the proxy server, thereby implementing native ABAC. We also plan to work on this in the future.



(a)



(b)

Figure 3: Time taken to enforce ABAC with varying number of users and (a) user attributes, (b) rules

ACKNOWLEDGMENTS

Research reported in this publication was supported by the National Science Foundation awards CNS-1564034, CNS-1624503 and CNS-1747728 and the National Institutes of Health awards R01GM118574 and R35GM134927. The work of Shamik Sural is supported by CISCO University Research Program Fund, Silicon Valley Community Foundation award number 2020-220329 (3696). The content is solely the responsibility of the authors and does not necessarily represent the official views of the agencies funding the research.

REFERENCES

[1] Batra, G., Atluri, V., Vaidya, J., & Sural, S. (2019). Deploying ABAC policies using RBAC Systems. Journal of computer security, 27(4), 483–506.
[2] Ene, A., Horne, B., Milosavljevic, N., Rao, P., Schreiber, R., & Tarjan, R. (2008). Fast exact and heuristic methods for role minimization problems. 2008 ACM Symposium on Access Control Models and Technologies (SACMAT).
[3] Colombo, P., & Ferrari, E. (2017). Towards a Unifying Attribute Based Access Control Approach for NoSQL Datastores. 2017 IEEE 33rd International Conference on Data Engineering (ICDE), 709-720.
[4] Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., & Abramov, J. (2011). Security Issues in NoSQL Databases. 2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications, 541-547.
[5] DB-Engine Rankings. Retrieved January 9, 2021 from https://db-engines.com/en/ranking
[6] MongoDB Official Documentation. Retrieved January 9, 2021 from https://docs.mongodb.com/.
[7] Hu, C., Ferraiolo, D.F., Kuhn, D.R., Schnitzer, A., Sandlin, K., Miller, R., & Scarfone, K. (2019). Guide to Attribute Based Access Control (ABAC) Definition and Considerations [includes updates as of 02-25-2019].
[8] Colombo, P., & Ferrari, E. (2017). Enhancing MongoDB with Purpose-Based Access Control. IEEE Transactions on Dependable and Secure Computing, 14, 591-604.