Low-code approach of Fine-Grained Access Control policies enforcement on data-centric applications

Hoang Phuoc Bao Nguyen[1,2], Manuel Clavel[2]

[1](Cohort 2022) Computer Science Department, ETH Zürich, Switzerland
[2]Computer Science Department, Vietnamese-German University, Vietnam

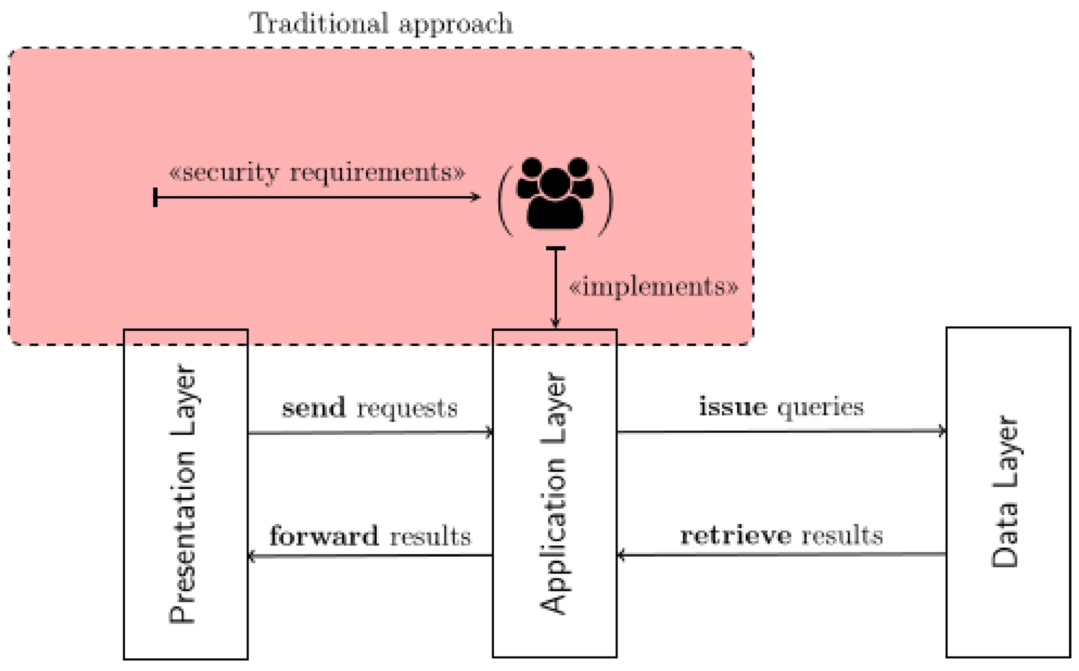
ETH zürich
VGU Vietnamese - German University

1. Introduction to the problem

Data-centric applications are focused around so-called **CRUD** actions that **C**reate, **R**ead, **U**pdate, and **D**elete (potentially sensitive) data.
Access control (AC) is a security mechanism that controls the conditions in which a *user* is allowed to perform *actions* over some data. Currently, most database-management systems offer almost **no standard support** for complex access control policies, as in the case of *fine-grained access control* policies. As a consequence, it is the responsibility of the development team to implement these policies.

Figure below describes how these policies are traditionally implemented:

1. authorization requirements are *collected* to the development team, then
2. the team *programmatically implements* these policies as functions in the application layer.
3. For every query requested, the application layer retrieves data from the lower layer and handles the access control checks.



This common practice, clearly, has some **drawbacks**:

1. Any compromise to the application layer results in exposure of the database system.
2. The application layer must perform (potentially complicated) checks, impacting negatively the overall performance of the application.
3. It is error-prone and furthermore, changes in the FGAC policy will necessarily imply non-trivial, ad-hoc changes in the application layer.

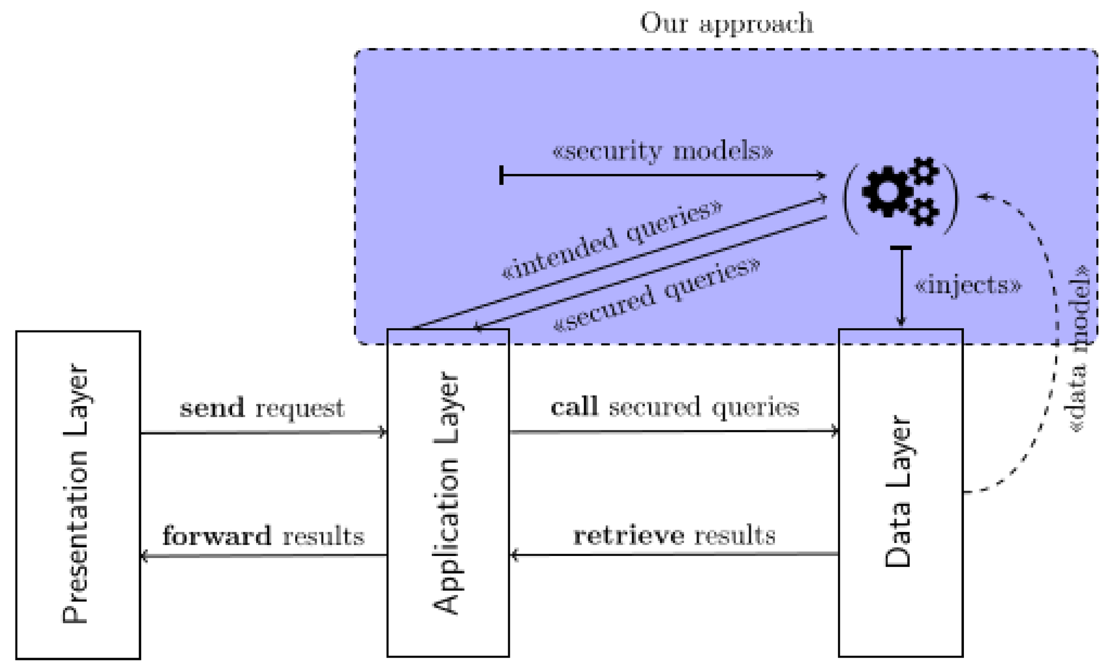
2. Our proposal and methodology

We propose to develop a **model-driven methodology for enforcing advanced policies on the data layer**. More specifically, we propose to develop a software component, which will

- take the authorization requirements (in a formal manner) and *automatically* generate corresponding executable authorization checks in terms of data layer functions, and
- for any (unsecured) data request actions, it *automatically* rewrites these actions into secured ones, by *calling* those aforementioned functions.

Figure below describes in a nutshell the basic workflow of our methodology:

1. authorization requirements are *formally defined* (i.e. security models), then
2. our component takes these requirements and *automatically generates* functions that perform the corresponding authorization checks.
3. For every SQL query, our component rewrites it into a secured version by injecting the aforementioned function calls.



Advantages of our approach:

1. privacy-sensitive data will not leave "uncontrolled" the database, not even for the purpose of performing authorization checks at the application layer.
2. FGAC checks will perform more efficiently at the data layer, levering on the highly sophisticated optimizations for filtering data .
3. The generated security artifacts are guaranteed to be *correct* and at the same time it is easy to adapt when the policies change.

3. State of our research

Our target. We focus on *relational* database and applications that use *relational* database management system (RDBMS, e.g., MySQL Community Server).

- **For modelling authorization requirements,** we use SecureUML and the Object Constraint Language (OCL),
  e.g. a lecturer *caller* is authorized to read a student *self* if he/she is the lecturer of that student can be expressed in OCL as an expression as:

$$\mathit{caller}.\texttt{students} \rightarrow \texttt{exists}(\texttt{s}|\texttt{s} = \mathit{self})$$

- For every OCL expression of an authorization policy, we require a corresponding SQL implementation of that OCL expression.
  e.g. the aforementioned OCL expression can be implemented in SQL as:

```
SELECT EXISTS (
  SELECT TRUE FROM Student_Lecturer
  WHERE students = self AND lecturers = caller)
```

- **For generating SQL authorization functions,** we rely on our notion of authorization enforcement in SQL queries.
- We implement our component as prototype and test it on a non-trivial case study.

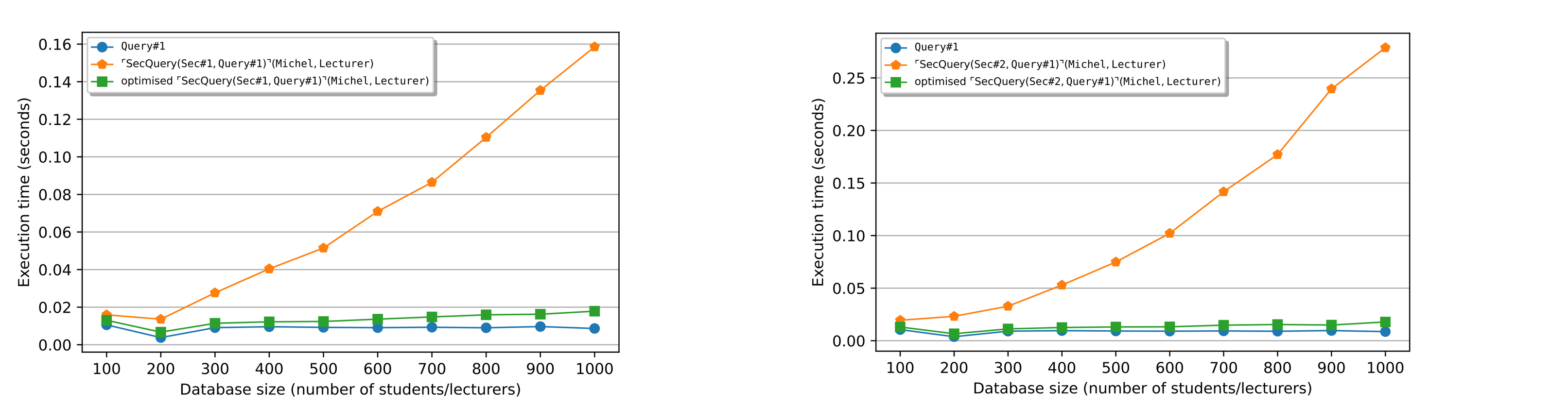


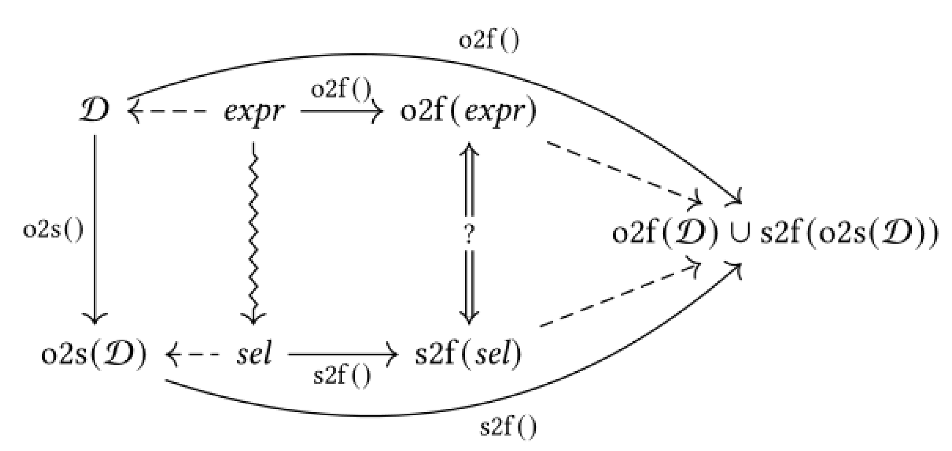
Figure 1: Preliminary experiment results of execution-time of our methodology. In a nutshell, the examples chosen here are of the same SQL query: *get the age of all students*, with different access control policies and over databases of different sizes. As the result have shown, executing the unsecured query (i.e., the blue line) will result as the fastest, executing the secured version of the query (i.e., the orange line) contains some performance overhead; and lastly, executing the secured version with some optimization that explained later (i.e., the green line) result "on par" w.r.t. the näive execution.

4. Refining our approach: Optimizing the secured queries

Problem. The secured queries that our methodology generated produces performance overhead.
Optimization. In several use cases, authorization checks are unnecessary (i.e. it always returns true in the given context) and therefore can be removed. We provide a formal methodology to assist checking whether these checks are required or not.

5. Proving correctness for SQL implementations of OCL expressions

As mentioned, our methodology requires an SQL implementation for every OCL authorization expression.



We presents a novel, SMT-based methodology of proving that an SQL implementation is *correct* w.r.t. an OCL expression. In a nutshell, we translate both SQL implementation and OCL expression to many-sorted first-order logic (MS-FOL) formulae and rely on the off-the-shelf Satisfiability Modulo Theories (SMT) solvers.

6. Future work

- **Applicability.** Provide an industrial case study with a complex underlying database with complex policies that showcase our approach.
- **Languages.** Extend our approach with different class of database systems (e.g., NoSQL), to support a wider range of SQL queries, and/or to support a larger class of OCL constraints.

References

[1] N. P. B. Hoang. Intelligent enforcement of fine-grained access control policies for sql queries. Master's thesis, Universidad Autonoma de Madrid, 2021.
[2] N. P. B. Hoang and Manuel Clavel. Model-based Characterization of fine-grained Access Control Authorization for SQL Queries. Journal of Object Technology, 19(3):3:1–13, 2020.
[3] N. P. B. Hoang and Manuel Clavel. A Model-Driven Approach for Enforcing Fine-Grained Access Control for SQL Queries. Springer Nature Computer Science, 2(5):370, 2021.